AR-010-284

DSTO

Software System Visualisation: Netmap Investigations

Peter Duffett and Rudi Vernik

DSTO-TR-0558

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Software System Visualisation: Netmap Investigations

Peter Duffett and Rudi Vernik

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0558

ABSTRACT

Defence systems have become increasingly reliant on software. The intangible and complex nature of software makes it difficult to manage and understand. Computer based visualisations of software have shown promise for providing the necessary visibility to acquire, develop, and maintain software systems. In this report we investigate a generic visualisation tool, Netmap, as a means of addressing these visualisation problems. Issues of using generic visualisation tools to support software tasks are discussed.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR No. AR-010-284
July 1997

APPROVED FOR PUBLIC RELEASE

Software System Visualisation: Netmap Investigations

Executive Summary

Defence systems have become increasingly reliant on software. The intangible and complex nature of software makes it difficult to manage and understand. Computer based visualisations of software have shown promise for providing the necessary visibility to acquire, develop, and maintain software systems.

This report provides results of investigations undertaken to assess the ability of a generic computer based visualisation tool, Netmap, to support software visualisation. Although focussing on visualisations to support software related tasks, the information presented would also be useful for those seeking to set up and use Netmap in other domains. Netmap usually requires additional tools (for example, parsers or metrics gathering tools) to extract data from the underlying source of information. The effort required to convert this data into a form suitable for using in Netmap can be considerable.

The results of these investigations show that Netmap's novel representations are useful for providing an overview or 'footprint' of a software system. An appreciation of a system's general characteristics, such as size and structural complexity, can be gained from a Netmap system footprint. Netmap is less useful for detailed analyses where access to a range of information types, such as found in CASE tools and many forms of software documentation, is required.

A discussion of the issues associated with using a generic visualisation tool for specific software tasks is presented. This report argues that beyond exploratory activities, Netmap would require additional features to effectively support task-based facilitation including guidance and attention direction.

Authors

Peter Duffett
Information Technology Division

Peter is currently working on the Advanced Visualisation and Description of Software (AViDeS) task which aims to research and transition visualisation and description techniques for software. He has a B.E. (Comp. Sys.) (Hons) from the University of Adelaide. He has been employed in the Software Engineering Group of the Information Technology Division of the DSTO since graduating in 1994.

Rudi Vernik
Information Technology Division

Rudi Vernik is a Senior Research scientist employed in Software Engineering Group, Information Technology Division. He currently leads the Advanced Visualisation and Description of Software (AViDeS) task. His research focuses on software systems visualisation, large-scale software engineering, and information logistics. Rudi has a Bachelor of Electronic Engineering (with Distinction) and a Diploma of Communications Engineering from the Royal Melbourne Institute of Technology and a PhD in Computer and Information Science from the University of South Australia.

Contents

1. INTRODUCTION **1**

1.1 Background **1**

1.2 Purpose **1**

1.3 Scope and Context **1**

1.4 Presentation **2**

2. OVERVIEW OF NETMAP **2**

2.1 Netmap Display Formats **3**

2.2 Grouping Strategy **4**

3. APPROACH **4**

3.1 Overview **4**

3.2 Software Engineering Data Used **5**

3.2.1 The Entities 5

3.2.2 The Relationships 6

3.2.3 Software Systems Studied 7

3.3 Setup Procedures **7**

3.3.1 Netmap Data Requirements 9

3.3.2 General Setup Procedure 9

3.3.3 The Netmap Scripts 10

3.3.4 Additional Setup Considerations 10

3.3.5 Summary of Problems encountered 11

4. RESULTS **12**

4.1 Viewing Overall Code Structure **12**

4.1.1 Overview of the Ada Composer system. 12

4.1.2 Overview of LIU 13

4.1.3 Overview of DGU 14

4.1.4 Overview of Ada SAGE 14

4.2 Viewing Subsystem Structure 14

4.2.1 Displaying SEE-Ada Subsystems with Netmap 15

4.2.2 Identifying Subsystems with Netmap 15

4.3 Viewing Compilation Unit *With* Dependencies 16

4.4 Viewing Encapsulation Relationships 16

5. DISCUSSION 16

5.1 Usefulness of Netmap Visualisations 17

5.1.1 The Netmap Display Format 17

5.1.2 Other Display Formats 17

5.1.3 Integration of Displays and Sources of Information 18

5.1.4 Applicability of Netmap for viewing software characteristics 18

5.2 Setup Considerations 19

5.2.1 Defining requirements 19

5.2.2 Accessing, Filtering and Integrating Information 19

5.2.3 Tailoring the Netmap Environment 19

5.3 Providing Task Support 20

5.3.1 Usage Guidelines 20

5.3.2 Recording Tool Usage 20

5.3.3 Facilitating Tool Use 20

6. CONCLUSIONS 21

7. REFERENCES 22

APPENDIX A - GENERATING DATA FILES 23

APPENDIX B - NETMAP.SCRIPT 25

APPENDIX C - MISCELLANEOUS SETUP FILES 27

APPENDIX D - NETMAP PLOTS 37

APPENDIX E - OVERVIEW OF SEE-ADA VERSION 3 48

E.1 Introduction 49

E.2 System Framework 49

E.3 SEE-Ada Views 50

E.4 Viewing Attribute Information 51

E.5 Viewing Relationships 52

E.6 Usage Monitoring 54

E.7 Script Mode 55

1. Introduction

1.1 Background

There are major barriers to viewing large software products. This is largely due to the fact that software is an intangible, synthetic and usually complex product. Moreover, software projects typically produce vast amounts of information. The sheer volume of this information makes it difficult to use. These factors can make acquiring, developing or maintaining software a complicated task.

This document reports on research conducted as part of the Advanced Visualisation and Description of Software (AViDeS) task (ALO 94/081) which is sponsored by the First Assistant Secretary Defence Materiel. The main objective of the AViDeS research is to address problems and difficulties being experienced in the acquisition, development and maintenance of large Defence software systems due to :

- poor project and product visibility; and
- the ineffectiveness of, and high costs associated with, the provision and use of many forms of project and product information (e.g. documentation, metrics).

The AViDeS work focuses on defining and exploring techniques which provide enhanced software product and project visibility through more effective use of underlying project information (eg as captured by integrated project support environments and tools). It is based on presenting information by way of computer-based visualisation techniques rather than through the production of vast amounts of software paperwork and metrics as is current practice for many large software projects.

1.2 Purpose

The purpose of this report is to document the results of a study which investigated the use of Netmap (release 3.54s3) (Davidson 1993), a commercially available database visualisation tool, as a basis for software visualisation. The report provides a detailed account of how Netmap was used to visualise the structural characteristics of several software systems.

1.3 Scope and Context

The data and information used in these investigations was obtained from SEE-Ada, a tool developed by DSTO's Information Technology Division (ITD). SEE-Ada is an integrated visualisation tool for Ada software which allows the access, filtering, customisation and display of software engineering information. SEE-Ada maintains its own database of this information which comprises data on software product structure as well as sets of related product, process and resource attributes (e.g. module size, complexity, configuration status and test results). The investigations discussed in this report used the structural information as a data source. The

SEE-Ada data was used since it provides a comprehensive and integrated source of software product information for a range of systems. More information about SEE-Ada is contained in Appendix E.

Readers of this report would benefit from a knowledge of the basics of the Ada programming language. Some basic software engineering knowledge would also aid understanding.

1.4 Presentation

The report is structured as follows:

- Section 2 provides an overview of the Netmap tool. It provides definitions for the terms that will be used when discussing Netmap characteristics.
- Section 3 discusses the approach used for the investigations. It describes the data used and the rationale for selecting the systems to be investigated. The section finishes with a record of how Netmap was set up to display SEE-Ada data and the problems encountered while doing this.
- Section 4 presents the results of the investigations on the four software systems studied.
- Section 5 provides a discussion of the findings of the study.
- Section 6 draws conclusions on the investigations and summarises the issues involved when using a generic visualisation tool such as Netmap for software visualisation.
- Appendix A,B and C contain useful scripts and data files for extracting SEE-Ada structural information and importing it into Netmap.
- Appendix D contains all of the Netmap plots which are discussed in this report.
- Appendix E provides a brief overview of SEE-Ada.

2. Overview of Netmap

Netmap is a commercial database visualisation and analysis tool that has found applications in many areas including fraud analysis, marketing, and communications (Davidson 1993). It can provide assistance in situations where there is a need to understand the inter-relationships between different classes of entities. Netmap can be used to view a variety of entities such as people, organisations, products, accounts, concepts, and requirements.

Netmap uses information sourced from one or more databases where common relationships exist, and presents the information through a variety of different views. It represents the database information as nodes (entities) and links (relationships).

- **Nodes.** A node is a logical entity. Nodes are characterised by the attributes that describe them. Each node attribute can have a value associated with it. For example, there might be a 'people' node that has the attribute 'hair colour' which in turn has the value 'brunette'.

- **Links.** Links are relationships between nodes. They are described by the qualifiers associated with them. For example, a link may exist between two 'people' nodes representing the telephone calls between the two individuals. The link qualifier might be the actual number of calls made between the two.

2.1 Netmap Display Formats

Netmap presents information by means of five different display layouts (or views). No examples of the Row/Column or the Bullseye view are provided in this report since they did not figure in the investigations presented. Further information on Netmap display layouts is available in the Netmap User's Manual (1994). The five display layouts are:

1. **Netmap Display.** This display is the predominant view used when displaying Netmap relationships. An example of a Netmap display is shown in Appendix D - Plot 1. Nodes are arranged in a circular fashion similar to spokes on a wheel. Node grouping is shown by adjoining nodes of the same group and having a space between node groups (Node grouping is discussed in Section 2.2). Links between nodes of different groups are drawn across the hub of the circle.

   Links between nodes of the same group are shown differently. To make their presence more obvious, satellites are employed. Satellites are the smaller circles which lie on the rim of the main node circle. Each satellite consists of all of the nodes in the corresponding group of nodes. Large groups therefore have correspondingly large satellites. Links between members of the group are shown within the circular satellites.

2. **Column Display.** A Column display arranges groups of nodes in columns from left to right. Satellites may be displayed above each of the columns to show intra-group links. No ordering is made within the group to simplify link connections across groups (i.e. the order of the nodes in a particular column is not optimised to reduce the number of link cross-overs). An example of a column view is shown in Plot 5.

3. **Row Display.** A Row display organises groups of nodes vertically from top to bottom, and group members horizontally left to right. Satellites may be displayed to the right of each of the rows to show intra-group links. An example of a row view with satellites displayed is shown in Plot 6.

4. **Row/Column Display.** The Row/Column display arranges groups horizontally in a row initially, then in a column at the second and subsequent levels.

5. **Bullseye Display.** The Bullseye display shows the first groups near the center of the figure, with each of the node's links radiating outward to the next group of nodes.

2.2 Grouping Strategy

The way nodes are arranged on a particular display is determined by the strategy employed for grouping nodes together. There are three grouping strategies used by Netmap:

1. **Pre-defined.** When the Pre-defined grouping strategy is selected, the nodes are grouped by a chosen attribute. Plot 1 provides an example of pre-defined grouping where nodes have been grouped by file.

2. **Emergent.** When the Emergent grouping strategy is selected, the nodes are grouped solely on the links present on the displayed map. Nodes are grouped if:

   - there are three or more nodes in a group,
   - each node has links to two or more nodes within the group, and
   - each node has at least half of its links to others within the group

   An example of an emergent grouping of package specifications and bodies based on *with* links is shown in Plot 8.

3. **Step Link.** When the Step Link grouping strategy is selected, nodes are grouped according to the minimum number of link traversals required to link back to a specified initial group, i.e. grouping all nodes that are the same number of links away from the initial group. Plot 9 shows step links from the node 'TEXT_IO'.

3. Approach

This section outlines the approach taken for the investigations discussed in this report. It begins by discussing the purpose and aims behind the investigations. The data used for the study is discussed and characterised in Section 3.2. Section 3.3 records how Netmap was setup to display SEE-Ada data and the problems encountered.

3.1 Overview

The purpose of this work was to explore the ways in which Netmap could be used to support Software Engineering (SE) tasks. The investigations presented in this report focus on visualising software structure.

The approach used for these investigations was to:

1. Define and select a subset of the available SEE-Ada information for analysis.

2. Import this information into Netmap.

3. Explore various Netmap visualisation approaches for viewing this information.

4. Consider how the visualisations may be used to support particular SE tasks.

5. Comment on the perceived general usefulness of Netmap for other types of SE information and tasks.

The specific aims of this study were to use Netmap to view the following software characteristics:

- overall code structure
- subsystem structure
- compilation unit dependencies
- encapsulation relationships

These characteristics are discussed in Section 3.2.

3.2 Software Engineering Data Used

3.2.1 The Entities

The data used in the Netmap investigations were sourced from the SEE-Ada database. In particular, the information extracted related to Ada source code and design structure (both entities and relationships). For an overview of these and other types of information available in the SEE-Ada database refer to the SEE-Ada Technical Reference (1996) Section 2.1.2.

Table 3-1 provides a list of the entities that are the focus of this study. These include Ada compilation units and subprogram units as well as the subsystem objects. The entities are represented as nodes within Netmap. These entities were chosen as they represent important items of interest to a software engineer and exercise Netmap with a range of node types.

Table 3-1 : Entities Used

Procedure Specification	Procedure Body	Procedure Subunit Specification	Procedure Subunit Body
Function Specification	Function Body	Function Subunit Specification	Function Subunit Body
Package Specification	Package Body	Package Body Subunit Specification	Package Body Subunit Body
Task Specification	Task Body	Task Body Subunit Specification	Task Body Subunit Body
Generic Package	Null Object	Physical Subsystem	Logical Subsystem

All but three of the objects are Ada units and are defined in The Reference Manual for the Ada Programming Language (1983). The following objects are SEE-Ada specific:

- **Null Objects** A Null Object is a compilation unit that has been referenced by other objects in the system, but whose source code has not been parsed. For example, an Ada library unit such as TEXT_IO would be classed as a null object.

- **Subsystem Objects** Subsystems are used within SEE-Ada as a mechanism for supporting abstraction at a higher level than the package. There are two types of subsystems:

  1. A **Physical Subsystem** represents groupings of Ada library units.

  2. A **Logical Subsystem** is a conceptual entity which can be used to encapsulate physical subsystems and other logical subsystems. This type of subsystem can be used to represent design abstractions.

The colours used for the entities in Table 3-1 can be found on the legend of the plots in Appendix D.

3.2.2 The Relationships

Table 3-2 shows the SEE-Ada relationships that were used in these investigations and represented in Netmap as links. These relationships constitute a range of important links between the chosen entities.

Table 3-2 : Relationships Used

Relationship Types	**Description**
Withs	The *with* relationship is a compilation dependency relationship defined within the Ada programming language. Relationships of this type connect units that depend on other units by virtue of them explicitly *withing* the other unit.
Spec Of	The relationship that exists between a compilation unit specification and the corresponding body.
Parent	The relationship between an encapsulating unit and the encapsulated unit.
Child Of	The relationship between a subunit and its parent unit.
Has Specification Unit	This relationship exists between a subunit and its specification unit.
Child Subsystem	The relationship of a logical subsystem to its child subsystems.
Library Unit	The relationship a physical subsystem has to its encapsulated library units.

Not all entities or relationships are shown on a particular plot, but rather a subset of each. The plots are discussed in detail in Section 4.

The colours used for the various relationships can be found on the legend of the plots in Appendix D.

3.2.3 Software Systems Studied

Four software systems provided the basis of the investigations discussed in this report. They were:

1. **Ada Composer**. Ada Composer is a graphical, object-oriented software design tool developed by Intermetrics Inc. It supports the interactive creation of Object Oriented Design Diagrams (OODD) and translates OODDs into compilable Ada Program Design Language.

2. **Launcher Interface Unit**. The LIU is a component of the Nulka system, a decoy system developed for use by naval ships against anti-ship missiles. The LIU software programs decoy rockets and initiates the launching process.

3. **Display Generator Unit**. The DGU is the embedded software system developed to provide navigation, weapons systems and surveillance functions for a military helicopter.

4. **Ada SAGE**. Ada SAGE is a Management Information System application development tool written at the Idaho National Engineering Laboratory. It is implemented as a set of Ada packages and a set of executable programs which are used as support utilities during application development and operation.

These four systems were chosen because they represent a cross section of various types of software systems. Each of the systems was written by a different organisation. The systems range from medium to large in size, providing a basis for assessing the scalability of Netmap.

Figure 3-1 shows some of the size characteristics of each of the four systems studied. For example, graph b) shows that the DGU and Ada SAGE systems are much larger (more Ada source lines of code (SLOC)) than the other two systems. Comparing a) and b) it can be seen that the average length of file is much smaller for the LIU than it is for the other systems. This characteristic is discussed more fully in section 4.1.2.

3.3 Setup Procedures

This section provides details of how Netmap was configured for analysis of SEE-Ada data. It first discusses the Netmap data requirements. This is followed by the setup procedure that was used to extract the SEE-Ada data and place it in a form suitable for use by Netmap. Scripts were developed to semi-automate the setup process. These are discussed in Section 3.3.3. The section concludes with a discussion of additional setup considerations and a summary of problems encountered during the setup process.

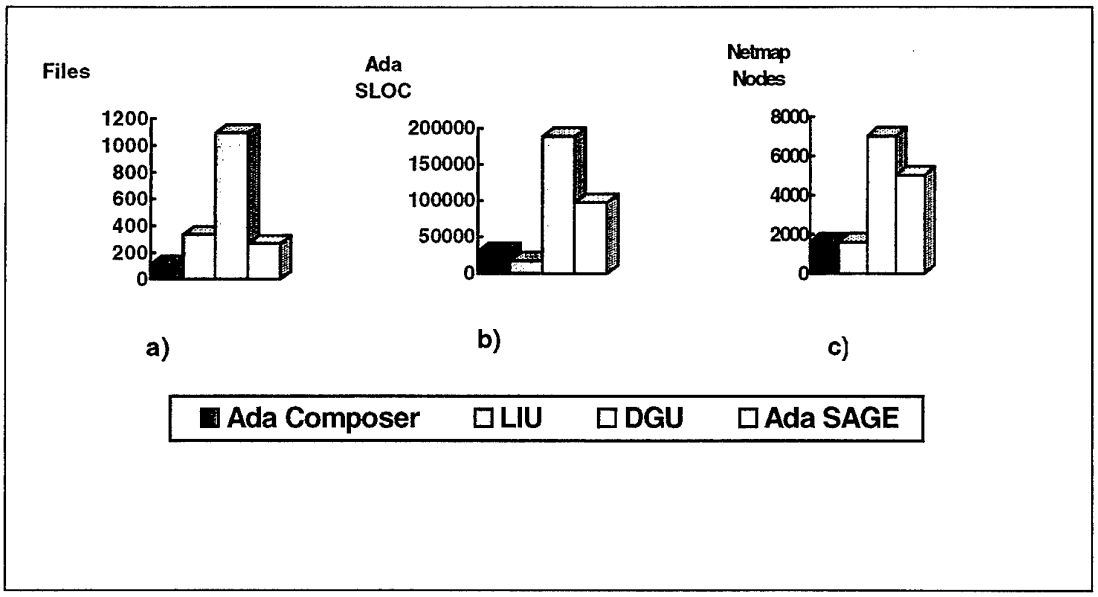


Figure 3-1 System Size Properties

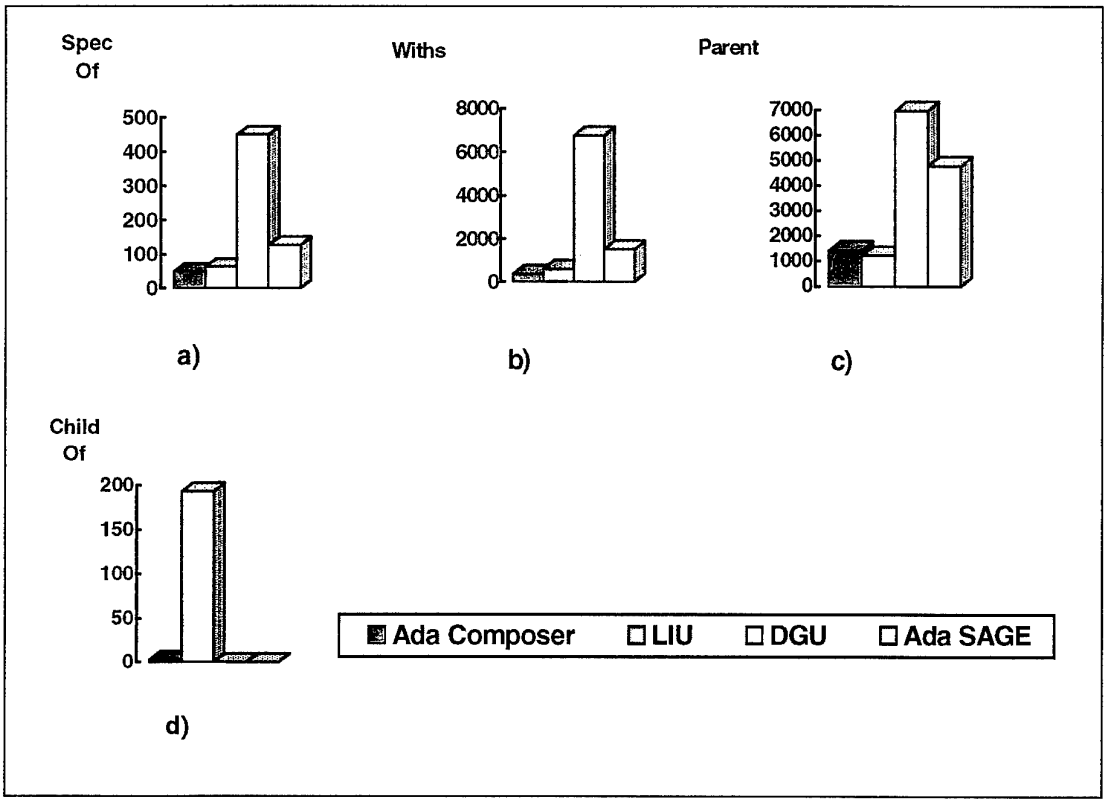


Figure 3-2 : System Relationship Statistics

3.3.1 Netmap Data Requirements

Netmap provides visualisation of data captured and imported into its database. A Netmap database comprises the following user supplied files:

- **Netmap Data Files.** These files contain the data on nodes, attributes, links and link qualifiers used by Netmap. The data must be in the form of ASCII text. The format must be of fixed width columns, comma delimited, or tab delimited text.

- **Data Layout Files.** For each Netmap data file, there needs to be (at least) one data layout file. This file describes the format of the data within the corresponding Netmap data file. (Refer to the Netmap Technical Manual (1994) Appendix C-16 for an explanation of layout commands and formats).

- **Code List Files.** Sometimes it is desirable to translate attribute and qualifier values appearing in the data into some other value, e.g. converting 1, 2, .., 7 into Sunday, Monday, .., Saturday. These files specify the translation. (Refer to the Netmap Technical Manual (1994) Appendix C-13 for additional information on code list files).

- **Database Definition File.** The DDF defines the label and the data type of each field of the new formatted database. This file is also used to specify the names of Code List Files to translate node attribute or link qualifier values. (Refer to the Netmap Technical Manual (1994) Appendix C-10 for more information).

3.3.2 General Setup Procedure

The following steps were performed to set up Netmap to display a SEE-Ada software system:

1. **Define the Data Requirements.** Decide what data is needed and identify the SEE-Ada database tables that hold this information. (The SEE-Ada database is implemented using the Oracle Relational Database Management System).

2. **Generate the Netmap data files from the SEE-Ada database.** Netmap data files can be generated by interfacing directly to the Oracle database using SQL*Plus. SQL*Plus establishes an Oracle connection and allows the user to interface to the SEE-Ada database tables through a SQL prompt (SQL Reference Manual 1990). The user then 'spools' the appropriate database tables as ASCII text to a file. To do this efficiently the page size has to be set to 0, and the line size must be large enough so that no carriage returns are inserted. Appendix A contains instructions for this process.

3. **Generate the Data Layout Files.** These files provide Netmap with a definition of the format of the files generated in step 2. For the SEE-Ada data, which is stored in fixed width columns, this involves calculating the first and last columns for the specific data field required within the data file.

4. **Produce a Single Data File.** Netmap uses only one data file. This file is constructed by first appending each layout to its corresponding data file and then concatenating all of the data-layout pairs together.

5. **Create the Database Definition File and Code List Files.** In this step, the Database Definition File is created as well as any code list files that may be needed. For SEE-Ada data it is desirable to have a number of code list files. This is because some of the required information (e.g. the compilation unit type) is stored in the Oracle database as a numeric identifier. The generation of the code list files removes the need to search through reference guides to decipher a node's attributes. Instructions on how to produce these files can be found in the Netmap Technical Manual (1994) Appendix C.

6. **Create a Formatted Netmap Database.** Once all of the necessary files are in place, the Netmap database can be created. Netmap comes with a utility program called 'CRDB' which performs this task. CRDB can be called from the directory containing the data files as follows:

```
>crdb -projdir ../ <system_name>
```

3.3.3 The Netmap Scripts

Performing the procedure described in Section 3.3.2 is quite tedious and repetitive when configuring several systems. It is also possible that the Netmap menus will be slightly different across systems due to such things as concatenating the data files together in a different order. Two scripts were developed to speed up the process and remove menu irregularities. Both of these scripts can be found in the appendices of this document together with a general description of each.

The first script, which can be found in Appendix A, uses a mixture of Unix and SQL commands to help generate the ASCII output files from the SEE-Ada database. The second script (Appendix B) is a Unix script. This script generates a Netmap database from the ASCII database dump and the standard configuration files for SEE-Ada data. Other files and filters were also used. These were the standard layout files discussed in Section 3.3.2 and a filter 'sqlfilter' used to cut superfluous text from the data files exported through SQL*Plus. All of these files can be found in Appendix C.

3.3.4 Additional Setup Considerations

The previous steps provided a general method for data extraction and formatting. The following additional setup considerations also needed to be taken into account:

- **Colour Setup** The default Netmap colour setting included only 8 colours. This was inappropriate for the data being examined which consisted of over 30 different node types. The Netmap documentation , (Netmap Technical Manual 1994) page 6-8, states that a maximum of 128 colours may be specified. The actual number may be less than this depending on hardware limitations. The supported colours were identified and a selection chosen to represent different node types.

Netmap reads the colour setting from the `colours.set` file and associates each colour with a node type. The colour given to a node type depends on its position in the declaration, i.e. the first node type declared is given the first colour in the list and so on. This is limiting in that if the colours for one attribute are specified through ordering of the colours in the colour settings file, the colours of all of the other attributes are determined. The colour settings file used for these investigations can be found in Appendix C.

The same problem does not exist for link colours since they can be individually assigned to link types through the linkmenu definition file (.lmf). (Refer to the Netmap Technical Manual (1994) Page 6-28).

- **Providing a Legend** Legends, such as the one shown on the bottom of Plot 1, are only available on printed output. There is no facility for displaying a legend of colour mappings on the screen. The file specifying the legend for printed output can be created and added to plots as desired. (Refer to the Netmap User's Manual (1994) Appendix C-2 for information on how to provide a legend). The legend file, 'Legend.key', used for our investigations can be found in Appendix C.

3.3.5 Summary of Problems encountered

A number of problems were encountered when setting up and using Netmap version 3.54s2. The problems listed here refer only to features that didn't work as specified, not to problems associated with trying to view software product structure.

- When creating a Netmap database, the code list files are read and used to create the attribute definition file (.adf) and the node menu file (.nm). It was found that should a '0' be entered as an attribute value, Netmap would increment this value and subsequent values. This had the effect of misaligning a value and its translated name from the code list file.

  This problem was avoided by correcting the '.adf' and the '.nm' files after they were generated by the CRDB utility. The correction was easier to perform if the attribute value '0' was positioned on the last line of the code list file so that only one attribute value needed to be corrected in each case.

- The first line of data in the data file was never correctly parsed. In our case this meant that the first compilation unit was always incorrectly interpreted. This cause the first node to have an incorrect ID and name. No workaround was found for this problem.

- Netmap could not deal with too many spaces occurring before a node ID.

  For example, " 5" could not be read in properly whereas " 5" could.

  If a node ID had too many preceding spaces, Netmap would assign its own node ID. To avoid this occurring the range of columns in the layout file needed to be reduced. This solution constrains the length of node names, and hence the number of nodes able to be included.

- When printing a postscript file containing plot information, the first colour defined in the file `color.set` was ignored. In its place the second colour defined in `color.set` would be assigned. All of the other colours printed correctly. This problem appears to be a problem of the postscript generator.

  Our solution to this problem was to manually change the postscript file generated by Netmap. Where it referenced the first colour (dark blue) in `color.set`, the reference was changed to the third colour, which was also dark blue.

4. Results

The results of these investigations focus on the visualisation of the structural characteristics of software engineering information. The results are organised in terms of the aim of the investigation as described in Section 3.1.

All of the plots referred to in this section can be found in Appendix D. Markers, e.g. (1), are used to indicate a position of interest on a plot.

4.1 Viewing Overall Code Structure

Our first goal in using Netmap was to get an overview of code structure, or structural 'footprint', for each of the four systems under investigation. This was considered important as it would provide a basis for comparison of the systems and provide a starting point for further investigations.

After experimenting with various Netmap layouts and groupings we decided upon a standard set of Netmap node and link settings which could be used across all of the systems. This standard setting displays all of the nodes defined in Table 3-1 except the two subsystem types. The nodes are grouped according to the files to which they belong and shown using the Netmap display. Links shown are Withs, Parent, and Child Of. Plots 1 through 4 all use the standard setting.

4.1.1 Overview of the Ada Composer system.

Plot 1 shows the Ada Composer system. The main circle shows all subprogram units (e.g. procedures, functions and tasks) and Ada compilation units (e.g. subunits, packages, generics, and main procedures) in the system grouped according to their encapsulating source file. It can be seen from the colouring of the node groups that some files contain specifications only, others contain bodies only, and a few contain specifications and bodies. Files in general contain a single compilation unit (e.g. (1) contains one package body and a number of subprograms) but there are some exceptions to this (e.g. (2) contains three package specifications). The practice of placing multiple compilation units within one source file is discouraged by many project coding standards, including the Software Productivity Consortium guidelines (SPC Guidelines 1991) which are generally accepted by the software industry.

Each satellite shows the units that are contained within a file. Links within satellites show subprogram unit encapsulation within a compilation unit. In most cases the software exhibits a simple encapsulation structure within a compilation unit.

Some files contain packages which encapsulate a large number of subprograms (those with large satellites containing many subprograms), while others encapsulate none. The plot also shows that there are package specifications which do not have any subprograms (these are type definition packages). The main procedure is identified (3) as a single procedure body. This plot highlights the fact that little use is made of generic packages, with only one being used (4). Null objects (5) are grouped together as they are all given the same default file ID.

The Netmap zoom feature can be used to get more detail on part of a Netmap view. Plot 10 shows the results of zooming in on the bottom right section of Plot 1. Plot 10 - (1) shows a more complex package structure where subprograms encapsulate other subprograms. Netmap plots of this kind can provide good insights into the structural complexity within compilation units.

The green links within the main circle show uni-directional *with* dependencies between units. The units which depend on large numbers of other units are easily identified by large numbers of converging lines. Incidences of high *with* concentrations could be further investigated if desired by either zooming in or profiling the nodes in question. Refer to the Netmap User's Manual (1994) pages 3-13 and 3-21 for more information about zooming and profiling.

The magenta links (child of) shows the connection between a subunit and its parent. There are only three such links in the Ada Composer system.

4.1.2 Overview of LIU

As Figure 3-1 shows, the LIU system is a similar size in terms of lines of code and number of nodes to the Ada Composer system. However, it has a different structure or 'footprint' as can be seen from Plot 2. Comparing Plot 1 with Plot 2 highlights some major differences between the two systems. Figure 3-1provides an indication as to why the plots look so different. Even though the LIU has almost the same number of nodes as the Ada Composer system, it has three times as many files, more *with* dependencies, and a large number of subunits. The high number of subunits is responsible for the large number of magenta links in the main circle. These structural characteristics can prove detrimental to system understanding and hence maintainability (Vernik and Landherr 1993).

It can be seen that the satellites of the LIU system are generally smaller than those for the Ada Composer system. This indicates that each file typically contains less subprogram units which infers that there is less structural design complexity within compilation units. Although there is less structural complexity within compilation units, there is a high degree of inter-unit complexity as evidenced by the large number of links between compilation units. This may make understanding unnecessarily difficult since an analyst may need to reference several files and maintain a mental map of dependencies to gain an understanding of software functionality. There are also implications for configuration management since the

system comprises a larger number of inter-related parts than may have been necessary.

The magenta (child of) links are much more apparent on Plot 2 and reflect the large number of subunits that are present in the LIU system.

Once again, only minor use has been made of generic packages, three being used.

4.1.3 Overview of DGU

Plot 3 shows a footprint of the DGU system. This system is significantly larger than the previous two systems examined (refer to Figure 3-1 for comparison). The Netmap for this system highlights this fact. It immediately indicates that the system comprises a large number of files containing compilations units with vast numbers of 'with' linkages.

There are few sizeable satellites. This could indicate that there is little design abstraction in compilation units. Further investigation, such as gaining an insight into the subprogram encapsulation structure, can be undertaken by zooming into satellites of interest.

Little use has been made of advanced Ada features such as generics and subunits.

4.1.4 Overview of Ada SAGE

The Ada SAGE system, as shown in Plot 4, has a similar number of nodes as the DGU, however their footprints are quite different. For example, the Ada SAGE system has significantly fewer *with* links. This would be expected given that the Ada SAGE system is a set of packages used as reusable components, not a single piece of application software. Moreover, there is significantly more encapsulation within the files.

A feature that becomes immediately apparent with the Ada SAGE system is the distinct grouping of the compilation units, with bodies appearing around the top of the main circle and specifications on the lower semi-circle. Although this feature captures the analyst's attention it is in reality a relatively insignificant one related to the way in which the information was captured by SEE-Ada. What has happened here is that all of the files containing bodies have been parsed into SEE-Ada after those containing specifications. This emphasises the fact that additional knowledge is required by the analyst to interpret features which Netmap highlights.

Marker (1) is next to a group of package specifications with one generic package. This is the file which was used to provide abstract data types. Unique features like this are difficult to quickly identify through direct viewing of the source code.

4.2 Viewing Subsystem Structure

Subsystems are used in SEE-Ada to represent design entities which support abstraction at a higher level than is provided by an Ada package. They are used to

capture and aid understanding of high level system structure. The subsystem objects used by SEE-Ada are described in Section 3.2.1.

4.2.1 Displaying SEE-Ada Subsystems with Netmap

Plot 5 shows the Ada Composer Subsystem using a Column Display. The column view was found to be the most appropriate representation for displaying tree-like structures such as the subsystem view. (Plot 5 can be compared to the SEE-Ada subsystem view which is shown at the top of Figure 7-2 in Appendix E). The plot was generated by using a step-link grouping strategy starting at the root of the tree. The root of the subsystem tree has to be determined before beginning this process.

A limitation of this representation is that a user cannot reorder the nodes within a group so that the links don't cross. Also, users cannot set the colours for subsystem types as these are determined from the colour settings chosen for compilation units.

4.2.2 Identifying Subsystems with Netmap

If the only available information about a software system is source code then the grouping of compilation units into logical subsystems is generally done via a predominantly manual reverse engineering process. SEE-Ada provides features which support this process. The use of the emergent grouping approach as used by Netmap showed promise as a means of supporting the reverse engineering process. The theory behind this was that there would be a large number of intra subsystem links and a low number of inter subsystem links. It was conjectured that logical subsystems could be identified through the emergent grouping algorithm grouping together those nodes which were tightly coupled. Plots 7 and 8 were developed to test this.

Plot 7 is a Netmap plot of the Ada Composer package specifications employing an emergent grouping strategy. Two major groups were identified ((1) and (2)). However, most package specifications did not *with* each other and therefore were grouped. Those package specifications not *withing* any of the packages in the main Netmap circle are shown in four columns at the bottom of the plot. Nodes which have no displayed links in a plot are called **isolates**.

The subsystems identified by Netmap bear only a slight resemblance to the manually identified subsystems in SEE-Ada. The units making up the group labelled (2) all came from the subsystem identified by a user of SEE-Ada as 'Components', however not all package specifications from the 'Components' SEE-Ada subsystem were identified by Netmap. The units making up the group labelled (1) came from three different subsystems and did not comprise the total of any of the SEE-Ada subsystems.

Plot 8 is a second attempt at identifying subsystems. It again shows an emergent grouping of the Ada Composer system, but this time package bodies are shown as well as package specifications. With the bodies included, the two main groups of Plot 7 ((1) and (2)) have expanded to include more units and are represented in Plot 8 as (2) and (3) respectively. A new group (1) has also been discovered. Most package specifications still remained ungrouped.

These results do not mean that the subsystems identified by Netmap are invalid but it suggests that the algorithm is different from that used by a software engineer. Netmap bases its grouping on connectivity while a software engineer bases his/her decisions on the functionality of the various packages (amongst other things).

The current version of Netmap (release 3.54s3) does not allow the user to change the algorithm on which the emergent grouping is based. The algorithm used for grouping was discussed in Section 2.2. The emergent grouping algorithm may have proved to be more useful for extracting design information had it been able to be changed or at least have its parameters adjusted.

4.3 Viewing Compilation Unit *With* Dependencies

Netmap was also used as a means of viewing compilation unit *with* dependencies. For example, Netmap could be used to identify units which (could) use a package's subprograms, types or variables. Plot 9 was produced as an example of how Netmap could be used for this purpose. It shows a Netmap arrangement based on the step-links algorithm commencing at the 3 o'clock position with TEXT_IO. The gap at one end of every green *with* relationship represents an arrow head indicating the direction of the relationship

Moving anti-clockwise, the first group encountered is made up of units which *with* the TEXT_IO package. The subsequent group consists of units which *with* units in the first group (unless they are already included in the first group). The links have been filtered (via the Link menu) so that links going in the other direction have been removed. This ensures that only units dependent on TEXT_IO (either directly or indirectly) are displayed.

4.4 Viewing Encapsulation Relationships

Netmap can be used to view encapsulation relationships using any one of the five possible display formats. Plot 10 - (1) marks a zoomed in view of a satellite of the Ada Composer system shown in Plot 1. The encapsulation relationships are represented as orange lines within the satellite.

Plot 6 shows a better view of the encapsulation of subprograms within the satellite displayed in Plot 10 - (1). This view is the Netmap row view with the root node at the top of the plot representing the package body. The lower down a subprogram body appears, the more highly nested (encapsulated) it is. The column view was found to be a more effective view for displaying information of this type due to its relative compactness.

5. Discussion

This section begins with a discussion of the usefulness of Netmap visualisations for viewing software characteristics, particularly software system structure. Setup issues that need to be taken into account when considering whether or not to use Netmap are presented. The section concludes with a discussion of how a general

visualisation tool such as Netmap can be used to support specific tasks. This is an important aspect to software visualisation systems that is often neglected.

5.1 Usefulness of Netmap Visualisations

Netmap provides a number of methods for visualising information. This section discusses the use of display formats and grouping strategies, as used for software system visualisation.

5.1.1 The Netmap Display Format

The key visualisation provided by Netmap is the Netmap display. This novel representation provides a comprehensible display with high information density. These properties make it ideal for gaining a quick overview of a system under investigation. A software system can be characterised with a one page plot such as Plot 1 which shows the system 'footprint' of the Ada Composer system. Plots such as this can be used to gain an appreciation of the different structural language constructs used within a system as well as the relationships between them. This means that effective comparisons can be quickly made between systems of a similar size.

Although useful for medium-sized systems, the approach used to produce a system footprint does not scale for larger systems. As Plot 3 shows, many of the features of the DGU are not discernible from the one page plot. Although Netmap provides features that allow a user to zoom-in on a section of a Netmap, as discussed in Section 5.1.2, this may result in a loss of context.

The Netmap display may have been used to support a greater range of tasks, such as design recovery, if the number of available grouping algorithms could have been extended. One algorithm that is present is the emergent grouping algorithm, which was introduced in Section 2.2 and further discussed in Section 4.2.2. This algorithm showed promise as a means of supporting the design recovery process but its full potential could not be realised, because of its inflexibility. For example, if the algorithm could be changed to group nodes that had 70% of their links within a group, instead of the enforced 50%, subsystems may have been more accurately identified.

While the Netmap display was useful for gaining an overview of a software system, more detailed analysis requires additional displays. Even though the Netmap display can be used to view a subset of the total system, often alternate displays can present the information in a better way. This issue is discussed in Section 5.1.2.

5.1.2 Other Display Formats

Netmap provides four display formats besides the Netmap display. The complete list together with a description of each can be found in Section 2.1. Of these, the column display was found to be the most useful for looking at design encapsulation and subprogram nesting within compilation units. The remaining display formats were either not novel or not found to be applicable for visualising software systems.

A problem common to all of the displays was that it was easy to lose context when zooming in and then panning around a display. No on-screen cues are given as to where the displayed screen fits into the entire display. This creates difficulties for the analyst as they would be likely to zoom-out of a display to check the context. Investigations may take longer and analysis errors could be introduced.

Netmap is limited to its five display formats. Analysts often require information to be presented in other ways. For example, the available display formats do not have charting or statistical capability of the type used in Figure 3-1 and Figure 3-2. Other graph layouts, textual representations and tabular data representation methods might also be useful. The five displays are also unable to display certain types of information as found in requirements and design documentation, test reports, and version control information. This means that for a complete analysis of a software system additional tools would be required.

5.1.3 Integration of Displays and Sources of Information

Software analysis often requires different perspectives of the available information depending on the task and the user. Netmap has features which can be used to quickly move between different displays of the same information. This is a particularly useful property of Netmap. For example, if a set of interesting nodes and links has been identified using a Netmap display, the user can then change to a Column display without having to re-identify the node and link set.

Due to the complex nature of large software systems there is a need to hide irrelevant information so as not to overload or distract the analyst. Netmap can hide information through its filtering and zooming functions. Nodes can be filtered based on an attribute of the node or whether it is highlighted or not.

Section 5.1.2 discussed the need for additional tools to completely analyse a software system. Ideally, Netmap would seamlessly integrate with these additional tools. This would allow an analyst to choose the best representation for information without losing context when moving across tool boundaries. To do this effectively, Netmap would require an API (Application Programming Interface) or scripting interface so that tools could communicate their state to each other.

5.1.4 Applicability of Netmap for viewing software characteristics

Besides Netmap's application in viewing code structure, as was reported in Section 4, Netmap could find use in the viewing of other types of software information. Due to its ability to represent code and design relationships and attributes succinctly, Netmap would lend itself to viewing other code characteristics such as call profiling and the relationship between test cases and corresponding application code.

Netmap would not be as useful for viewing attribute values associated with a set of structural entities. This is due to Netmap's inability to simultaneously represent multiple attributes graphically. Information of this type includes version control information and software product measures such as size and complexity. The lack of statistical visualisation features was discussed in Section 5.1.2.

Netmap can, however, represent all of the attributes of a single node through its Node Profile function. A node profile shows a node's ID number, name, attributes and the currently displayed links. Links can be similarly profiled. A useful feature of the Node Profile function is the ability to place information directly onto a node during analysis. This allows a user to, for example, record analysis results about a node for later reference. Refer to the Netmap User's Guide (1994) Section 3-21..25 for a more detailed description of Node Profiling.

5.2 Setup Considerations

The initial overhead when configuring Netmap to a new source of information can be significant. The following section outlines the setup issues which need to be considered when using Netmap.

5.2.1 Defining requirements

The first step in carrying out a particular task (for example, software evaluation) is to define the requirements of the task. Requirements can be specified in terms of a set of goals to be achieved. Several approaches have been suggested for identifying information needs for specific goals (Rombach 1991), (Vernik 1996). These approaches help identify information needs by defining a set of questions that need to be answered for each goal. Only once these questions and information needs have been identified can the most appropriate Netmap visualisation be selected for the individual performing the task.

5.2.2 Accessing, Filtering and Integrating Information

All of the information used by Netmap has to be derived from other sources, be it in a database, text file or some other form. Netmap cannot directly access this information. Information filters or parsers are required to extract the information from its source and convert it into a form suitable for Netmap. No parsers or filters are provided with Netmap.

To correctly import the information, a detailed knowledge of the Netmap tool and various text processing tools is required. The import process is a non-trivial, time consuming exercise. Batch files or scripts would need to be developed for cost effective application of Netmap in a particular domain. The files and scripts which were developed for this report can be found in Appendices A, B and C.

5.2.3 Tailoring the Netmap Environment

Netmap is a general tool and it can present information in a multitude of ways. The features that are prominent are very dependent on the configuration of the Netmap environment. To use Netmap effectively requires a detailed knowledge of Netmap and an understanding of the domain of investigation. Netmap needs to be tailored so as to emphasise only those features which will assist in satisfying the investigation requirements.

Although Netmap provides some features for customising displays, it does not allow itself to be tailored as might be required for particular user tasks. Colour

legends and the ability to change the shape of nodes are available on Netmap printed plots through the Netmap Presentation Tool, but not interactively on the screen. It was also impractical to tailor the colour scheme depending on the attribute being used in the investigation as discussed in Section 3.3.4. Customising displays based on specific task definitions is discussed in Section 5.3.3.

5.3 Providing Task Support

Netmap is an example of a generic visualisation tool which has proved useful for exploratory activities. There are a number of issues that need to be considered if a tool such as this is to provide direct task support. This section identifies some of these issues.

5.3.1 Usage Guidelines

The most asked question when using Netmap was 'What is this telling me?'. There was no guidance to suggest which combinations of displays and node groupings would be appropriate for different types of situations. Effective use in a particular domain would require a set of guidelines for the use and interpretation of the Netmap views. There would ideally be a catalogue of domain specific tasks which would guide the user through a series of visualisations with features to look for and give the rationale behind each task.

5.3.2 Recording Tool Usage

Netmap would benefit from the addition of an automated recording tool which would record a user's interactions with Netmap. The tool could record, for example, the type of task the user was performing, the display formats and grouping strategies employed, zooming and panning operations, and the results found. Such a tool would help ensure that investigations were repeatable, a necessary condition for auditing work.

Recording tool usage provides other benefits such as a means of process and tool improvement (Phillips and Vernik, 1997) (Recker, 1994). The output logs produced could also be used to facilitate tool usage through the generation of tool automation scripts as discussed in Section 5.3.3.

5.3.3 Facilitating Tool Use

An important aspect of the usability of a visualisation tool is its external control mechanisms (Price, Baecker et al. 1993). Scripting control is one means for providing consistent and stable representations for well defined tasks such as software evaluations (Vernik 1996). This approach may also increase productivity. Currently Netmap does not have scripting facilities, which makes it best suited to exploratory tasks. With scripting facilities and a set of standard evaluation scripts, the Netmap tool could better support software engineering activities such as software product evaluations.

Visualisations such as those provided by Netmap can be used to direct the attention of the operator to particular features through the separation, colour and grouping of

nodes. The key is to make sure that the features which stand out are the features which are relevant to the task at hand. During the investigations discussed in this report, unimportant features often became prominent when using Netmap. This is to be expected given the generality of the tool. Netmap would benefit from an attention direction assistant which would automatically configure the display to support the current task. Prototypes of this kind of tool have been developed to support diagnostic radiology (Rogers 1995). Once again, scripting features could help facilitate this aspect of visualisation.

6. Conclusions

This report provides examples of how Netmap can be set up and used to view software product structure. The use of Netmap's novel visualisation techniques was investigated as a means of viewing software code and design structure. Their usefulness in gaining visibility of software products and support of software-related tasks has been discussed.

Netmap provides visualisations which are suitable for generating system footprints. These high-level views are suitable for gaining a understanding of the structural characteristics of a system. The generic nature of the tool together with the high information density of the Netmap view means that a wide range of systems can be quickly compared and contrasted at a high level.

There are significant setup issues which need to be considered when configuring Netmap to a new source of information. A mechanism is required for getting the underlying information into a form that can be best used by Netmap. This will usually involve using additional tools (for example, parsers or metrics gathering tools). Substantial effort may be required to develop tools to convert and integrate the data from the underlying source of information into a form suitable for analysis through Netmap.

Netmap may not be as useful as other tools (for example, SEE-Ada) for supporting software related tasks such as analysis and evaluation of software products. This is because Netmap is a general tool, not tailored to the software domain. However, certain Netmap visualisations are helpful in the initial exploratory work of software analysis. The way nodes and links are represented and laid out highlights areas of potential interest thereby reducing analysis time. Care has to be taken as prominent features may be unimportant.

The Netmap visualisations can trigger hypotheses about a particular system but these often have to be investigated outside of the Netmap environment due to the limited number of visualisations that Netmap supports. The lack of integration with other tools and absence of a powerful scripting interface means that Netmap is unable to provide customised software descriptions.

Netmap is a partial solution to problems of software system visibility and may be considered as a supplement to other evaluation tools.

7. References

Davidson, C. (1993). "What your Database Hides Away." New Scientist Jan 1993.

Netmap User's Guide (1994). "The User's Guide for The NETMAP System." Netmap Solutions Pty Ltd, North Sydney NSW. 1994

Netmap Technical Manual (1994). "The Technical Manual for The NETMAP System." Netmap Solutions Pty Ltd, North Sydney NSW. 1994

SQL Reference Manual (1990). "SQL Reference Manual, Oracle V6", Oracle Corporation.

Phillips and Vernik (1997). "Capturing and Analysing Usage of Computer-Based Tools". Defence Science Technology Organisation 1997. Yet to be published.

Price, B. A., R. M. Baecker, et al. (1993). "A Principled Taxonomy of Software Visualisation." Journal of Visual Languages and Computing **4**(3): 211-266.

Recker, M.M. (1994). "A methodology for analyzing students' interactions within educational hypertext." Educational Multimedia and Hypermedia, 1994

Rogers, E. (1995). "Cognitive Cooperation through Visual Interaction." Knowledge-Based Systems **8**(2): 117-125.

Rombach, H. D. (1991). "Practical Benefits of Goal-Oriented Measurement." in Software Reliability and Metrics (eds. Fenton N., Littlewood B.) Elsevier Applied Science: Chap 14.

SEE-Ada Technical Reference (1996). "SEE-Ada version 3 Technical Reference Manual." Defence Science Technology Organisation 1996.

SPC Guidelines (1991). "Ada Quality and Style: Guidelines for Professional Programmers." Software Productivity Consortium, 2214 Rock Hill Road, Herndon, Va.

Reference Manual for the Ada Programming Language (1983). "Reference Manual for the Ada Programming Language." American National Standards Institute.

Vernik, R. J. (1996). PhD Thesis "Visualisation and Description in Software Engineering." Computer and Information Science. Adelaide, University of South Australia 1996

Vernik, R. J. and S. F. Landherr (1993). "Lessons Learned from Quality Evaluations of Nulka Software." Defence Science and Technology Organisation 1993. ERL-0761-RE.

Appendix A - Generating Data Files

This appendix contains a generic method for extracting SEE-Ada database information and producing it in a form suitable for Netmap. Text enclosed within brackets <>, are arguments that need to be substituted with appropriate information. It is assumed that a directory called Netmap exists and that all systems are generated within subdirectories of the Netmap directory. Within the Netmap directory the file **netmap.script** must be present. The contents of this file are described in Appendix B.

```
-------------------------UNIX COMMANDS-----------------------

cd <Netmap system directory>
sqlplus <Oracle user name>/<Oracle user password>

-----------------------ORACLE COMMANDS-----------------------

set pagesize 0;
set FEEDBACK OFF

spool <system_name>_COMP_UNIT.dat
set linesize 153
select
IDENT,ADANAME,FILE_IDENT,TYPE,LINE,IS_LIBRARY_UNIT,HAS_SUBUNI
TS from <system_name>_COMP_UNIT;

spool <system_name>_COMP_UNIT_BODY.dat
set linesize 21
select IDENT,BODY_IDENT from <system_name>_COMP_UNIT where
BODY_IDENT is not NULL;

spool <system_name>_DEP.dat
set linesize 25
select * from <system_name>_DEP;

spool <system_name>_ENCAPS.dat
set linesize 124
select IDENT,ADANAME,FILE_IDENT,TYPE,LINE from
<system_name>_ENCAPS;

spool <system_name>_ENCAPS_PARENT.dat
set linesize 23
select IDENT,PARENT_IDENT from <system_name>_ENCAPS where
PARENT_IDENT <> 0;

spool <system_name>_SS_COMP.dat
set linesize 104
select SUBSYS_IDENT,SUBSYS_NAME,IS_LOGICAL from
<system_name>_SS_COMP;

spool <system_name>_SS_DESC.dat
set linesize 175
select * from <system_name>_SS_DESC;

spool <system_name>_SS_LOGI_CONT.dat
set linesize 26
select * from <system_name>_SS_LOGI_CONT;
```

```
spool <system_name>_SS_PHYS_CONT.dat
set linesize 27
select * from <system_name>_SS_PHYS_CONT;

spool <system_name>_SUBUNITS.dat
set linesize 139
select IDENT,ADANAME,FILE_IDENT,TYPE,LINE,HAS_DEPENDENCY from
<system_name>_SUBUNITS;

spool <system_name>_SUBUNITS_PARENT.dat
set linesize 23
select IDENT,PARENT_IDENT from <system_name>_SUBUNITS;

spool <system_name>_SUBUNITS_SPEC.dat
set linesize 21
select IDENT,SPEC_IDENT from <system_name>_SUBUNITS;
spool off;

------------------------UNIX COMMANDS-----------------------

cd Netmap
netmap.script <system_name>
```

Appendix B - netmap.script

This Unix script automates the process of setting up a Netmap database given the set of data files produced as described in Appendix A. It also uses a number of standard files which can be found in Appendix C.

```
#!/bin/csh
# Script to generate a .dat file from the ORACLE files
generated
# P. Duffett

if ($#argv != 1) then
  echo 'Usage : netmap.script <system_name>'
  exit 1
endif

set SYSTEM_NAME = $argv[1]
cd $SYSTEM_NAME

foreach I ( _COMP_UNIT _COMP_UNIT_BODY _DEP _ENCAPS
_ENCAPS_PARENT _SS_COMP _SS_DESC _SS_LOGI_CONT _SS_PHYS_CONT
_SUBUNITS _SUBUNITS_PARENT _SUBUNITS_SPEC )
  set FILE_NAME = {$SYSTEM_NAME}{$I}'.dat'
  set TEMP_FILE_NAME = {$SYSTEM_NAME}{$I}'.tmp'
  /users/pld/Utilities/sqlfilter $FILE_NAME $TEMP_FILE_NAME
  rm $FILE_NAME
  cat '../'{$I}'.layout' $TEMP_FILE_NAME > $FILE_NAME
  rm $TEMP_FILE_NAME
end

cat {$SYSTEM_NAME}_COMP_UNIT.dat {$SYSTEM_NAME}_ENCAPS.dat
{$SYSTEM_NAME}_SS_COMP.dat {$SYSTEM_NAME}_SUBUNITS.dat
{$SYSTEM_NAME}_COMP_UNIT_BODY.dat {$SYSTEM_NAME}_DEP.dat
{$SYSTEM_NAME}_ENCAPS_PARENT.dat {$SYSTEM_NAME}_SS_DESC.dat
{$SYSTEM_NAME}_SS_LOGI_CONT.dat
{$SYSTEM_NAME}_SS_PHYS_CONT.dat
{$SYSTEM_NAME}_SUBUNITS_PARENT.dat
{$SYSTEM_NAME}_SUBUNITS_SPEC.dat > {$SYSTEM_NAME}.dat

cp ../STANDARD.ddf {$SYSTEM_NAME}.ddf
cp ../*.cl .
crdb -projdir ../ $SYSTEM_NAME
cp ../STANDARD.adf {$SYSTEM_NAME}.adf
cp ../STANDARD.nm {$SYSTEM_NAME}.nm
cp ../STANDARD.lmf {$SYSTEM_NAME}.lmf
cp ../Legend.key .
```

Appendix C - Miscellaneous Setup Files

This appendix contains the standard files and script used for configuring SEE-Ada data for use in Netmap. The script utility, called 'sqlfilter', filters out the SQL commands still contained in the data files. The standard files consist of the legend (Legend.key), the colour setting file (colours.set), the attribute definition file (STANDARD.adf), the node definition file (STANDARD.ddf), the link menu definition file (STANDARD.lmf), the node menu file (STANDARD.nm), the layout files (*.layout) and the code list files (*.cl). These files can be re-used for all Netmap systems displaying SEE-Ada data. Descriptions of these file types can be found in the Netmap Technical Reference (1994).

sqlfilter

```
#include <stdio.h>
#include <string.h>

#define MAXSTRING  200

main(argc, argv)
int argc;
char *argv[];
{
  FILE *ifp, *ofp;
  char s[MAXSTRING];
  int done;

  if (argc != 3) exit(1);

  ifp = fopen(argv[1], "r");
  ofp = fopen(argv[2], "w");

  for(;feof(ifp)==0;) {
    fgets(s,MAXSTRING,ifp);
    if ((feof(ifp)==0) && (s[0]!='S') && (s[1]!='Q') &&
(s[2]!='L')) fputs(s,ofp);
  }
}
```

Legend.key

```
                                Legend
%x

          Unit Types                                    Link
Types

  %00b Procedure Specification                  %12b Spec Of
  %01b Procedure Body                           %28b Withs
  %02b Procedure Subunit Specification          %16b Parent
  %03b Procedure Subunit Body                   %03b Child
Subsystem
  %06b Function Specification                   %05b Library
Unit
  %07b Function Body                            %18b Child Of
  %08b Function Subunit Specification           %02b Has
Specification Unit
  %09b Function Subunit Body
```

```
    %12b Package Spec
    %13b Package Body
    %16b Package Body Subunit Specification
    %17b Package Body Subunit Body
    %18b Task Specification
    %19b Task Body
    %20b Task Body Subunit Specification
    %21b Task Body Subunit Body
    %28b Generic Package
    %29b Null Object

  %x
```

colours.set

```
  !white     { background }
  black     { foreground }
  white

  blue
  cyan
  blue
  violet
  yellow
  yellow
  blue
  cyan
  blue
  violet
  yellow
  yellow
  red
  pink
  yellow
  yellow
  orange
  tan
  magenta
  plum
  magenta
  plum
  yellow
  yellow
  yellow
  yellow
  yellow
  yellow
  green
  maroon
```

STANDARD.adf

```
  1 0 999 Subsystem Type
  2 0 999 Description
  3 0 999 File ID
  4 0 999 Unit Type
  5 0 999 Start Line
  6 0 999 Library Unit
  7 0 999 Subunits
  8 0 999 Dependency
  0
  1 0 Physical Subsystem
```

```
1 1 Logical Subsystem
4 0 Procedure Specification
4 1 Procedure Body
4 2 Procedure Subunit Specification
4 3 Procedure Subunit Body
4 4 Procedure Instantiation
4 5 Procedure Rename
4 6 Function Specification
4 7 Function Body
4 8 Function Subunit Specification
4 9 Function Subunit Body
4 10 Function Instantation
4 11 Function Rename
4 12 Package Spec
4 13 Package Body
4 14 Package Instantiation
4 15 Package Rename
4 16 Package Body Subunit Specification
4 17 Package Body Subunit Body
4 18 Task Specification
4 19 Task Body
4 20 Task Body Subunit Specification
4 21 Task Body Subunit Body
4 22 Task Type
4 23 Task Object
4 24 Generic Procedure Specification
4 25 Generic Procedure Body
4 26 Generic Function Specification
4 27 Generic Function Body
4 28 Generic Package
4 29 Null Object
6 0 Isn't Library Unit
6 1 Is Library Unit
7 0 No Subunits
7 1 Has Subunits
8 0 No Dependency
8 1 Has Dependency
0
```

STANDARD.ddf

```
%NODE_DEFINITION
   %NODE_ID
   %NODE_NAME
   %attribute subsystem_type %label "Subsystem Type"
%data_type int %code_list subsystem_type.cl
   %attribute description %label "Description" %data_type
str
   %attribute file_id %label "File ID" %data_type int
   %attribute unit_type %label "Unit Type" %data_type int
%code_list unit_types.cl
   %attribute start_line %label "Start Line" %data_type int
   %attribute library_unit %label "Library Unit" %data_type
int %code_list library_unit.cl
   %attribute has_subunits %label "Subunits" %data_type int
%code_list subunits.cl
   %attribute has_dependency %label "Dependency" %data_type
int %code_list dependency.cl

%LINK_DEFINITION
```

```
   %QUALIFIER relationship %label "Relationship" %data_type
str
```

STANDARD.lmf

```
%PROFILE : Profile
NODE_HIGHLIGHT : Highlighted Nodes
NODE_NAME IN [?] : Node Name
NODE_ID IN [?]: Node ID

%LINK2 Q2 : flags = c default_colour = sequential :
Relationship
1 : color = red : Spec Of
2 : color = green : Withs
3 : color = orange : Parent
4 : Child Subsystem
5 : Library Unit
6 : color = magenta : Child Of
7 : Has Specification Unit

%OUTPUT_FORMAT QDF
Q1 : INT : LINK_DIRECTION
Q2 : STR : Relationship
%WIDTH
1 : $1 : Number of Links

! add further options in the format:
! Data Source : Variable name & calculations : Label
! eg
! 1 | Q2 : $2/$1 : Average Value
! refer to Technical Guide for further details.


%DRAW_IF : Draw if Total
1 : $1 : Number of Links
! Add further options as per the Width section


!%LINK_FILTER : Filter Links on
! Add options in the same way to use Link Filtering.
! (uncomment the %LINK_FILTER line).
```

STANDARD.nm

```
%GROUP_MENU : Group nodes by
    %ENTRY : Subsystem Type
      NODE_GROUP = A1
    %ENTRY : Description
      NODE_GROUP = A2
    %ENTRY : File ID
      NODE_GROUP = A3
    %ENTRY : Unit Type
      NODE_GROUP = A4
    %ENTRY : Start Line
      NODE_GROUP = A5
    %ENTRY : Library Unit
      NODE_GROUP = A6
    %ENTRY : Subunits
      NODE_GROUP = A7
    %ENTRY : Dependency
      NODE_GROUP = A8
```

```
%COLOUR_MENU : Colour nodes by
    %ENTRY : Subsystem Type
      NODE_COLOUR = A1
    %ENTRY : Description
      NODE_COLOUR = A2
    %ENTRY : File ID
      NODE_COLOUR = A3
    %ENTRY : Unit Type
      NODE_COLOUR = A4
    %ENTRY : Start Line
      NODE_COLOUR = A5
    %ENTRY : Library Unit
      NODE_COLOUR = A6
    %ENTRY : Subunits
      NODE_COLOUR = A7
    %ENTRY : Dependency
      NODE_COLOUR = A8

%ORDER_MENU : Order nodes by
    %ENTRY : None
    %ENTRY : Subsystem Type
      NODE_ORDER = A1
    %ENTRY : Description
      NODE_ORDER = A2
    %ENTRY : File ID
      NODE_ORDER = A3
    %ENTRY : Unit Type
      NODE_ORDER = A4
    %ENTRY : Start Line
      NODE_ORDER = A5
    %ENTRY : Library Unit
      NODE_ORDER = A6
    %ENTRY : Subunits
      NODE_ORDER = A7
    %ENTRY : Dependency
      NODE_ORDER = A8
    %ENTRY : Number of links
      NODE_ORDER = NODE_NLINKS

%WIDTH_MENU : Base width on
    %ENTRY : None
    %ENTRY : Subsystem Type
      NODE_WIDTH = A1
    %ENTRY : Description
      NODE_WIDTH = A2
    %ENTRY : File ID
      NODE_WIDTH = A3
    %ENTRY : Unit Type
      NODE_WIDTH = A4
    %ENTRY : Start Line
      NODE_WIDTH = A5
    %ENTRY : Library Unit
      NODE_WIDTH = A6
    %ENTRY : Subunits
      NODE_WIDTH = A7
    %ENTRY : Dependency
      NODE_WIDTH = A8
    %ENTRY : Number of links
      NODE_WIDTH = NODE_NLINKS
```

```
%EXCLUDE_MENU : Exclusions
  A1=0 : Physical Subsystem
  A1=1 : Logical Subsystem
  A4=0 : Procedure Specification
  A4=1 : Procedure Body
  A4=2 : Procedure Subunit Specification
  A4=3 : Procedure Subunit Body
  A4=4 : Procedure Instantiation
  A4=5 : Procedure Rename
  A4=6 : Function Specification
  A4=7 : Function Body
  A4=8 : Function Subunit Specification
  A4=9 : Function Subunit Body
  A4=10 : Function Instantation
  A4=11 : Function Rename
  A4=12 : Package Spec
  A4=13 : Package Body
  A4=14 : Package Instantiation
  A4=15 : Package Rename
  A4=16 : Package Body Subunit Specification
  A4=17 : Package Body Subunit Body
  A4=18 : Task Specification
  A4=19 : Task Body
  A4=20 : Task Body Subunit Specification
  A4=21 : Task Body Subunit Body
  A4=22 : Task Type
  A4=23 : Task Object
  A4=24 : Generic Procedure Specification
  A4=25 : Generic Procedure Body
  A4=26 : Generic Function Specification
  A4=27 : Generic Function Body
  A4=28 : Generic Package
  A4=29 : Null Object
  A6=0 : Isn't Library Unit
  A6=1 : Is Library Unit
  A7=0 : No Subunits
  A7=1 : Has Subunits
  A8=0 : No Dependency
  A8=1 : Has Dependency
node_id in [?] : ID Code
node_name in [?] : Node name

%CC_TARGETS : Display all
      : nodes
  A1=0 : Physical Subsystem
  A1=1 : Logical Subsystem
node_highlight : highlighted nodes

!%CC_NODES  : linked to
!      : nodes
!  A1=0 : Physical Subsystem
!  A1=1 : Logical Subsystem
!node_highlight : highlighted nodes

%STEP_LINKS : Step Links
  A1=0 : Physical Subsystem
  A1=1 : Logical Subsystem
node_highlight : highlighted nodes
node_name in [?] : Node Names
```

_COMP_UNIT.layout

```
%BEGIN_LAYOUT COLUMN
%BEGIN_NODE
        %NODE_ID 7:10
        %NODE_NAME 12:91
        %file_id 96:102      %DEFAULT -1
        %unit_type 112:113
        %start_line 120:124  %DEFAULT -1
        %library_unit 140:140
        %has_subunits 153:153
        %subsystem_type 0    %DEFAULT -1
%END_NODE
%END_LAYOUT
```

_COMP_UNIT_BODY.layout

```
%BEGIN_LAYOUT COLUMN
  %BEGIN_LINK
        %NODE_ID 7:10
        %NODE_ID 18:21
        %relationship 0 %DEFAULT "Spec Of"
  %END_LINK
%END_LAYOUT
```

_DEP.layout

```
%BEGIN_LAYOUT COLUMN
  %BEGIN_LINK
        %NODE_ID 9:12
        %NODE_ID 22:25
        %relationship 0 %DEFAULT "Withs"
  %END_LINK
%END_LAYOUT
```

_ENCAPS.layout

```
%BEGIN_LAYOUT COLUMN
%BEGIN_NODE
        %NODE_ID 7:10
        %NODE_NAME 12:91
        %file_id 96:102      %DEFAULT -1
        %unit_type 112:113
        %start_line 120:124  %DEFAULT -1
        %subsystem_type 0    %DEFAULT -1
%END_NODE
%END_LAYOUT
```

_ENCAPS_PARENT.layout

```
%BEGIN_LAYOUT COLUMN
  %BEGIN_LINK
        %NODE_ID 7:10
        %NODE_ID 20:23
        %relationship 0 %DEFAULT "Parent"
  %END_LINK
%END_LAYOUT
```

_SS_COMP.layout

```
%BEGIN_LAYOUT COLUMN
%BEGIN_NODE
        %NODE_ID 9:12
        %NODE_NAME 14:93
        %unit_type 0   %DEFAULT -1
```

```
        %subsystem_type 104:104
  %END_NODE
  %END_LAYOUT
```

_SS_DESC.layout

```
  %BEGIN_LAYOUT COLUMN
  %BEGIN_NODE
        %NODE_ID 9:12
        %description 14:80  %DEFAULT "None"
  %END_NODE
  %END_LAYOUT
```

_SS_LOGI_CONT.layout

```
  %BEGIN_LAYOUT COLUMN
    %BEGIN_LINK
        %NODE_ID 10:13
        %NODE_ID 23:26
        %relationship 0 %DEFAULT "Child Subsystem"
    %END_LINK
  %END_LAYOUT
```

_SS_PHYS_CONT.layout

```
  %BEGIN_LAYOUT COLUMN
    %BEGIN_LINK
        %NODE_ID 9:12
        %NODE_ID 24:27
        %relationship 0 %DEFAULT "Library Unit"
    %END_LINK
  %END_LAYOUT
```

_SUBUNITS.layout

```
  %BEGIN_LAYOUT COLUMN
  %BEGIN_NODE
        %NODE_ID 7:10
        %NODE_NAME 12:91
        %file_id 96:102      %DEFAULT -1
        %unit_type 112:113
        %start_line 120:124
        %has_dependency 139:139
        %subsystem_type 0    %DEFAULT -1
  %END_NODE
  %END_LAYOUT
```

_SUBUNITS_PARENT.layout

```
  %BEGIN_LAYOUT COLUMN
    %BEGIN_LINK
        %NODE_ID 7:10
        %NODE_ID 20:23
        %relationship 0 %DEFAULT "Child Of"
    %END_LINK
  %END_LAYOUT
```

_SUBUNITS_SPEC.layout

```
  %BEGIN_LAYOUT COLUMN
    %BEGIN_LINK
        %NODE_ID 7:10
        %NODE_ID 18:21
        %relationship 0 %DEFAULT "Has Specification Unit"
    %END_LINK
```

```
%END_LAYOUT
```

dependency.cl

```
1 : Has Dependency
0 : No Dependency
```

library_unit.cl

```
1 : Is Library Unit
0 : Isn't Library Unit
```

subsystem_tye.cl

```
1 : Logical Subsystem
0 : Physical Subsystem
```

subunits.cl

```
1 : Has Subunits
0 : No Subunits
```

unit_types.cl

```
1  : Procedure Body
2  : Procedure Subunit Specification
3  : Procedure Subunit Body
4  : Procedure Instantiation
5  : Procedure Rename
6  : Function Specification
7  : Function Body
8  : Function Subunit Specification
9  : Function Subunit Body
10 : Function Instantation
11 : Function Rename
12 : Package Spec
13 : Package Body
14 : Package Instantiation
15 : Package Rename
16 : Package Body Subunit Specification
17 : Package Body Subunit Body
18 : Task Specification
19 : Task Body
20 : Task Body Subunit Specification
21 : Task Body Subunit Body
22 : Task Type
23 : Task Object
24 : Generic Procedure Specification
25 : Generic Procedure Body
26 : Generic Function Specification
27 : Generic Function Body
28 : Generic Package
29 : Null Object
0  : Procedure Specification
```

Appendix D - Netmap Plots

This appendix contains the Netmap plots together with a brief description of each.

PLOT 1. The Ada Composer system showing the unit types specified in the legend. The units are grouped according to the files to which they belong and arranged using the Netmap view. Links shown are Withs, Parent, and Child Of.

PLOT 2. The LIU system with the same configuration as Plot 1.

PLOT 3. The DGU system with the same configuration as Plot 1.

PLOT 4. The Ada SAGE system with the same configuration as Plot 1.

PLOT 5. A column view of the Ada Composer subsystem structure.

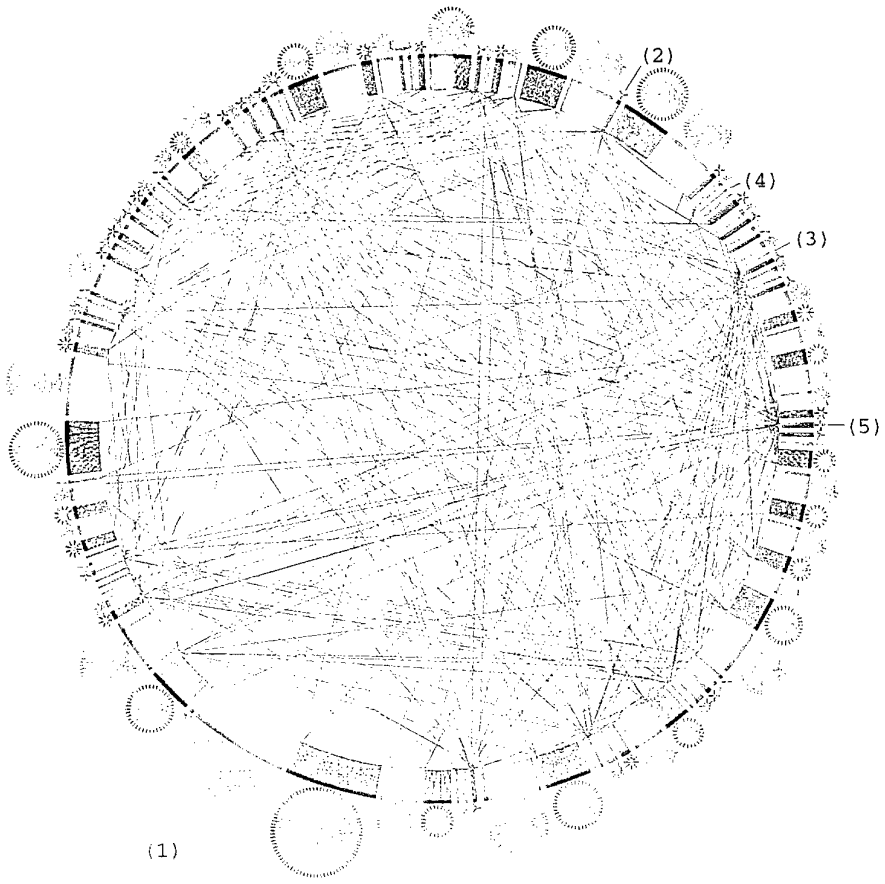
PLOT 6. A plot of step links from an Ada Composer package body shown in row view. The plot shows the package body at the top with differing levels of encapsulation of the procedure and function bodies below.

PLOT 7. A plot of the Ada Composer package specifications and the *with* links between them. The package specifications are grouped according to the emergent grouping algorithm.

PLOT 8. Similar to Plot 7, but this plot also shows Ada Composer package bodies.

PLOT 9. This plot shows the step links from a library unit (TEXT_IO) with all of the units that are directly or indirectly dependent on it. (Ada Composer System, Netmap view)

PLOT 10. A zoomed in view on the bottom right portion of Plot 1.



Legend

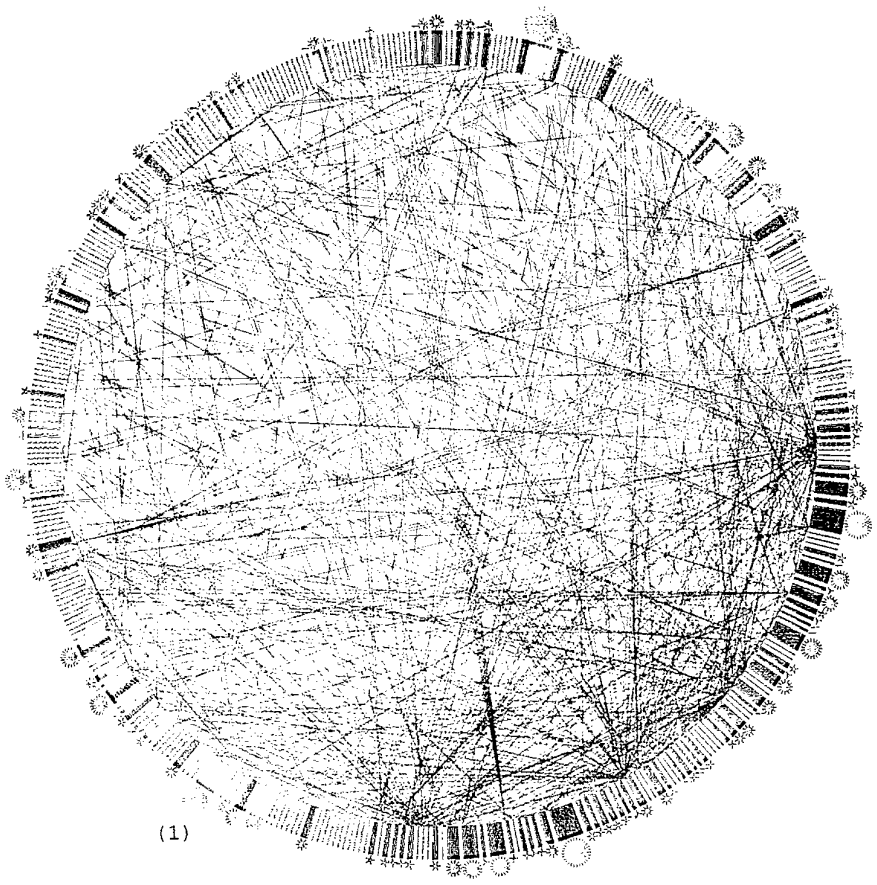
Unit Types

- Procedure Specification
- Procedure Body
- Procedure Subunit Specification
- Procedure Subunit Body
- Function Specification
- Function Body
- Function Subunit Specification
- Function Subunit Body
- Package Spec
- Package Body
- Package Body Subunit Specification
- Package Body Subunit Body
- Task Specification
- Task Body
- Task Body Subunit Specification
- Task Body Subunit Body
- Generic Package
- Null Object

Link Types

- Spec Of
- Withs
- Parent
- Child Subsystem
- Library Unit
- Child Of
- Has Specification Unit

Plot 1 Ada Composer - Netmap Layout



Legend

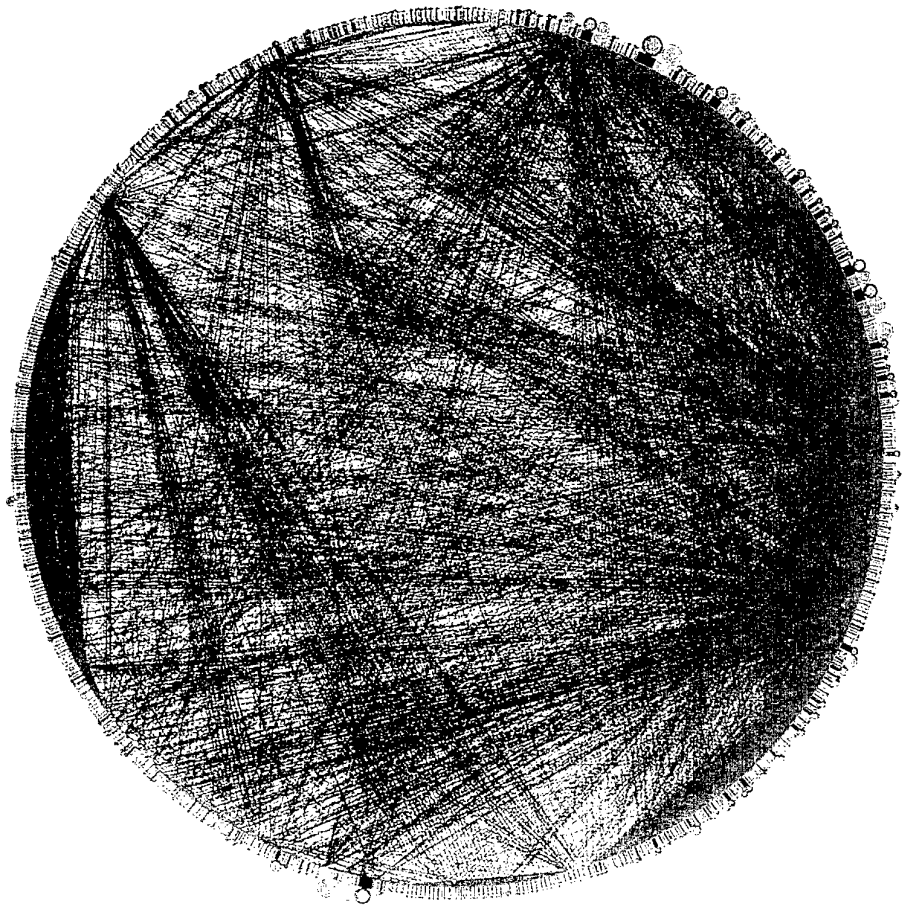
Unit Types

- Procedure Specification
- Procedure Body
- Procedure Subunit Specification
- Procedure Subunit Body
- Function Specification
- Function Body
- Function Subunit Specification
- Function Subunit Body
- Package Spec
- Package Body
- Package Body Subunit Specification
- Package Body Subunit Body
- Task Specification
- Task Body
- Task Body Subunit Specification
- Task Body Subunit Body
- Generic Package
- Null Object

Link Types

- Spec Of
- Withs
- Parent
- Child Subsystem Library Unit
- Child Of
- Has Specification Unit

Plot 2 LIU - Netmap Layout



Legend

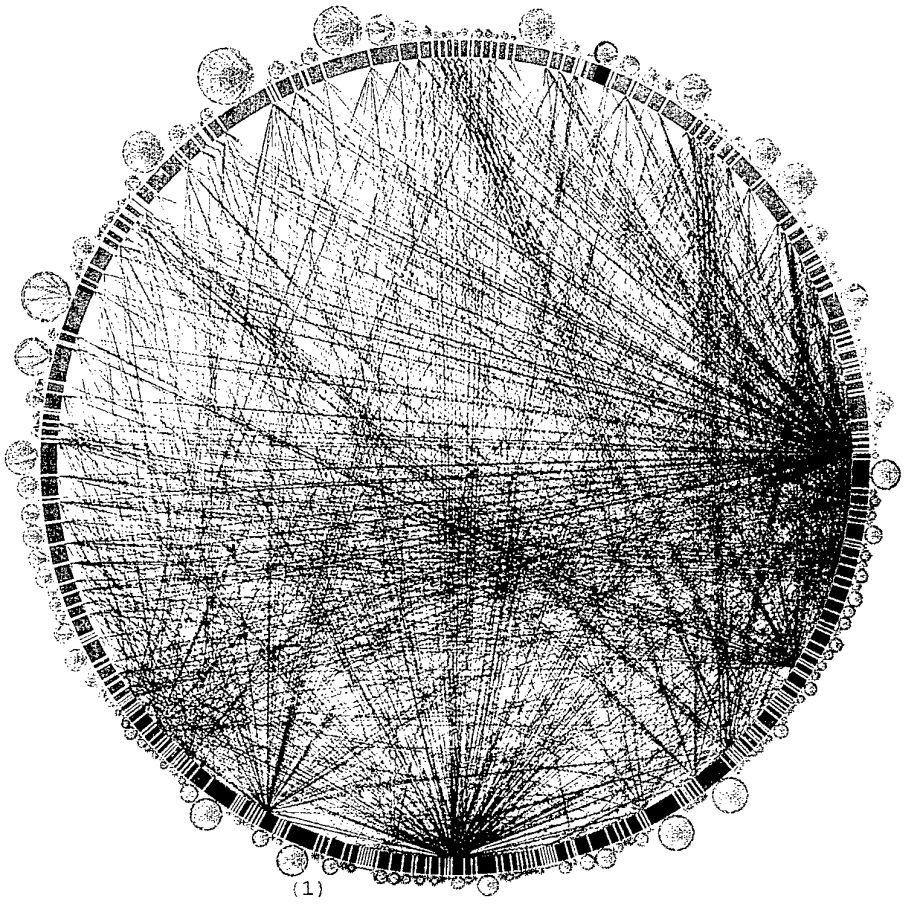
Unit Types

- Procedure Specification
- Procedure Body
- Procedure Subunit Specification
- Procedure Subunit Body
- Function Specification
- Function Body
- Function Subunit Specification
- Function Subunit Body
- Package Spec
- Package Body
- Package Body Subunit Specification
- Package Body Subunit Body
- Task Specification
- Task Body
- Task Body Subunit Specification
- Task Body Subunit Body
- Generic Package
- Null Object

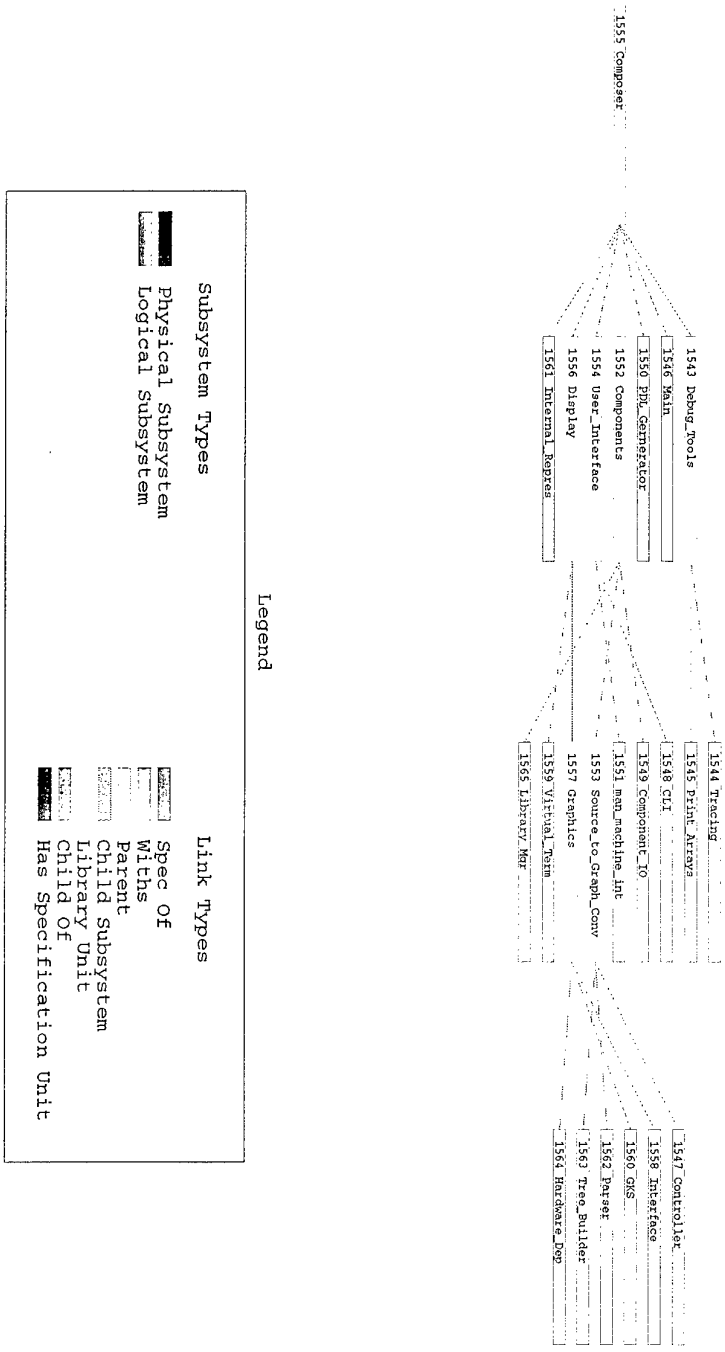
Link Types

- Spec Of
- Withs
- Parent
- Child Subsystem Library Unit
- Child Of
- Has Specification Unit

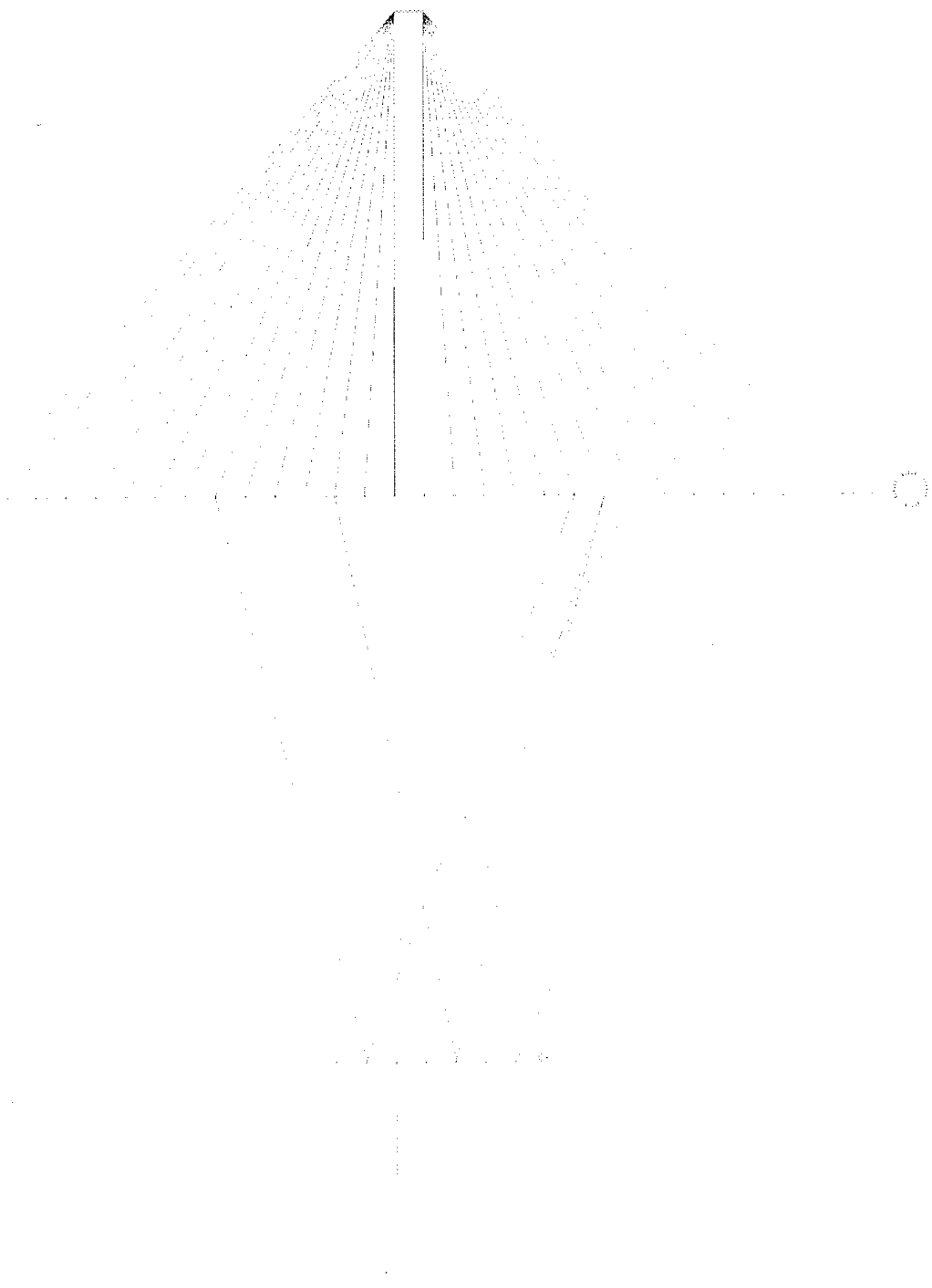
Plot 3 DGU - Netmap Layout



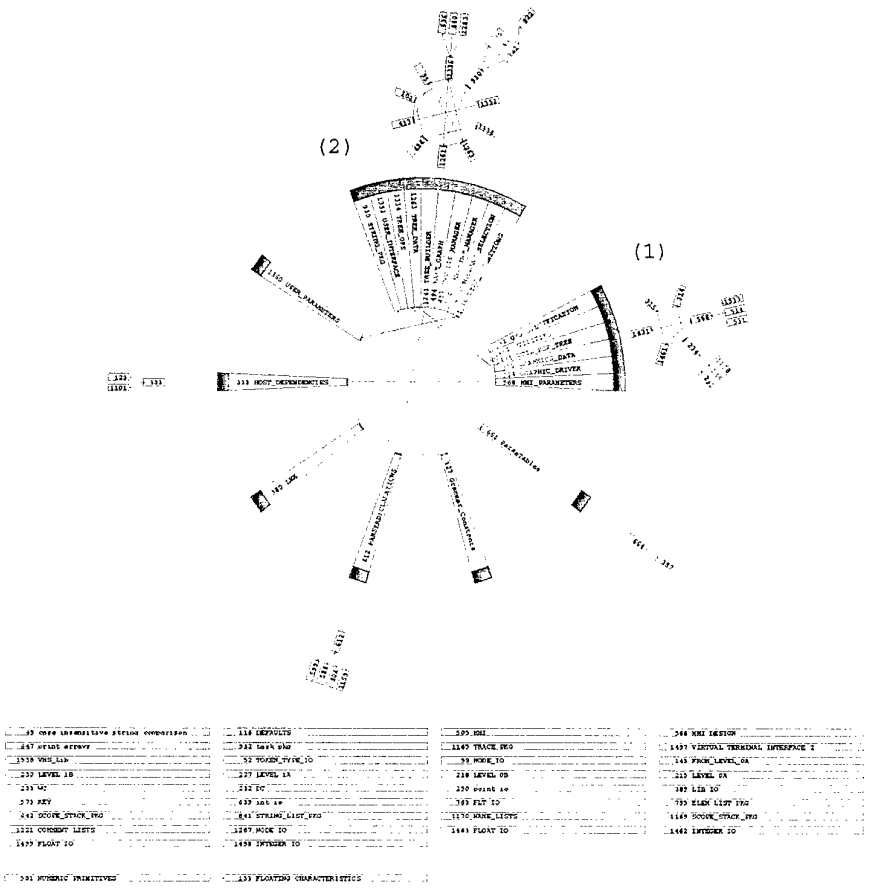
Plot 4 Ada SAGE - Netmap Layout



Plot 5 Ada Composer Subsystem - Column Layout



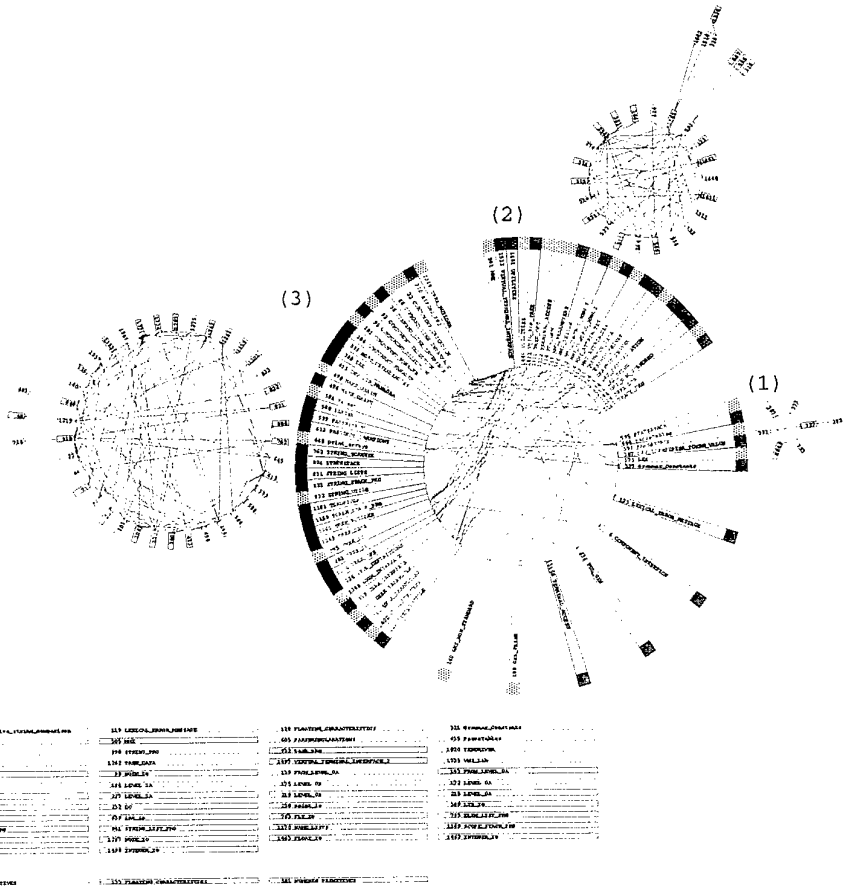
Plot 6 Ada Composer - Row Layout - Step Link Grouping



Legend

Unit Types	Link Types
Procedure Specification	Spec Of
Procedure Body	Withs
Procedure Subunit Specification	Parent
Procedure Subunit Body	Child Subsystem
Function Specification	Library Unit
Function Body	Child Of
Function Subunit Specification	Has Specification Unit
Function Subunit Body	
Package Spec	
Package Body	
Package Body Subunit Specification	
Package Body Subunit Body	
Task Specification	
Task Body	
Task Body Subunit Specification	
Task Body Subunit Body	
Generic Package	
Null Object	

Plot 7 Ada Composer - Emergent Grouping



Legend

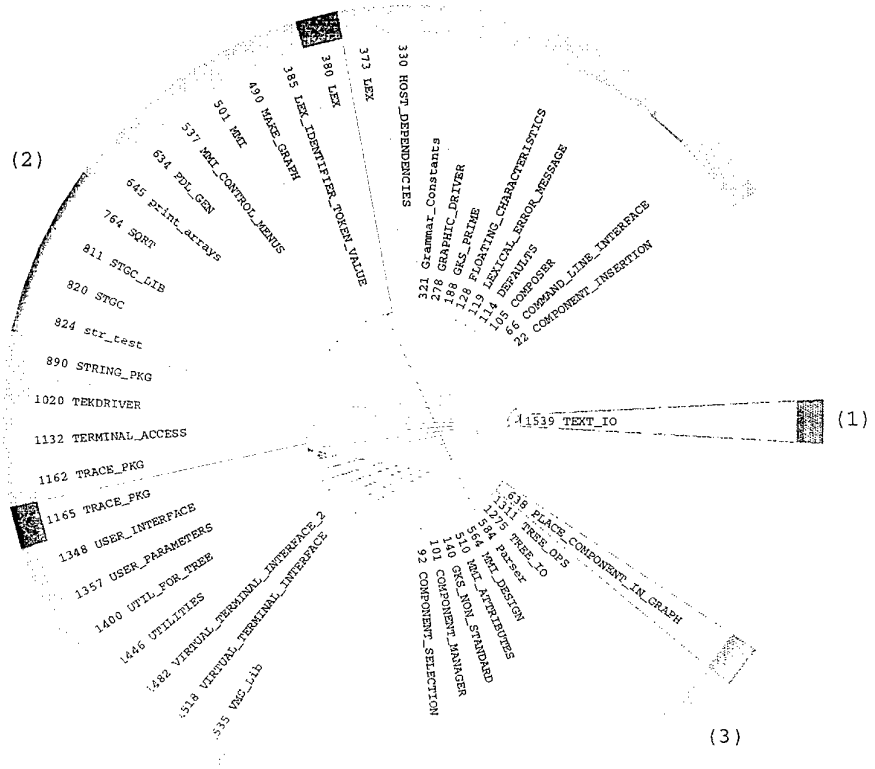
Unit Types

- Procedure Specification
- Procedure Body
- Procedure Subunit Specification
- Procedure Subunit Body
- Function Specification
- Function Body
- Function Subunit Specification
- Function Subunit Body
- Package Spec
- Package Body
- Package Body Subunit Specification
- Package Body Subunit Body
- Task Specification
- Task Body
- Task Body Subunit Specification
- Task Body Subunit Body
- Generic Package
- Null Object

Link Types

- Spec Of
- Withs
- Parent
- Child Subsystem
- Library Unit
- Child Of
- Has Specification Unit

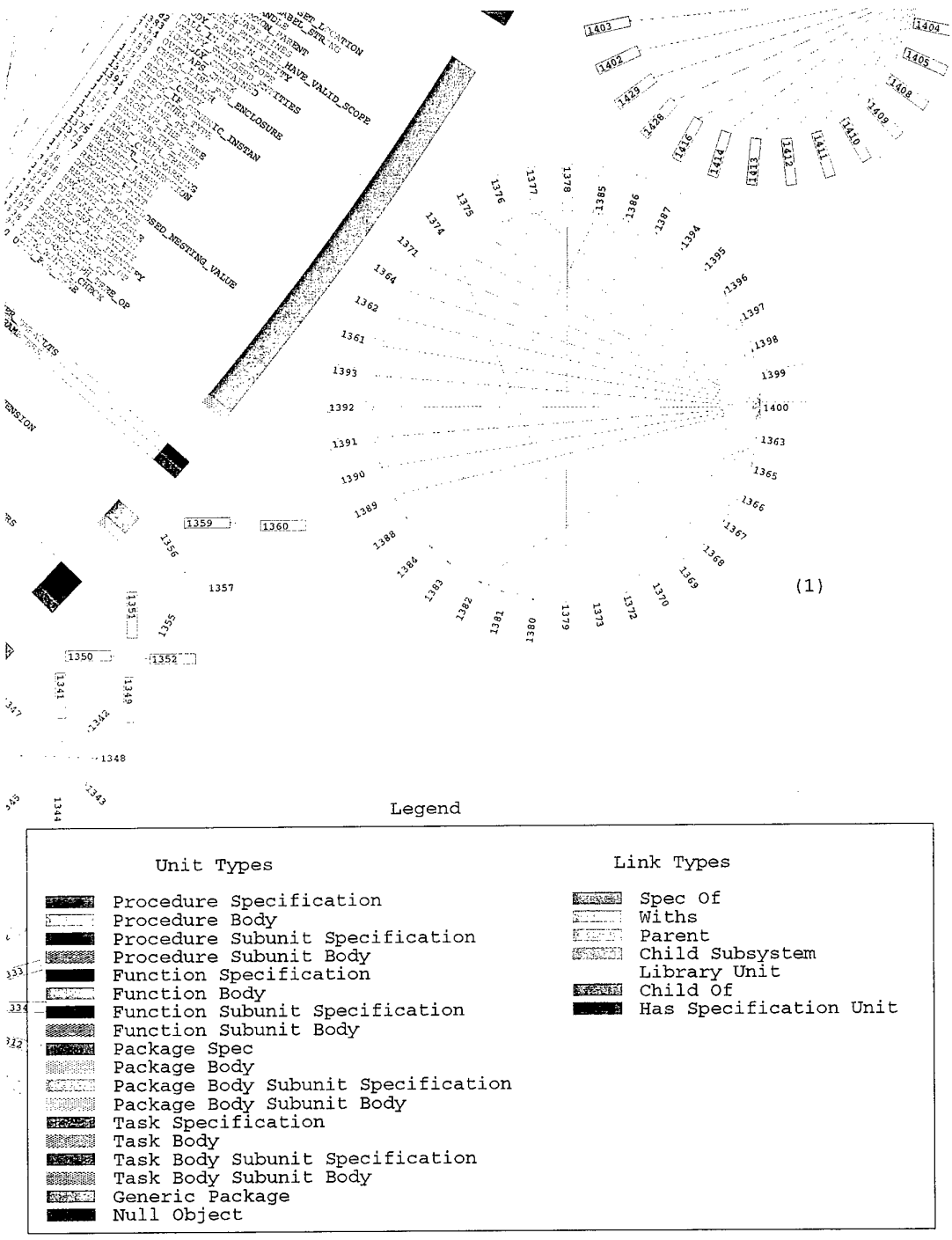
Plot 8 Ada Composer - Emergent Grouping



Legend

Unit Types	Link Types
Procedure Specification	Spec Of
Procedure Body	Withs
Procedure Subunit Specification	Parent
Procedure Subunit Body	Child Subsystem
Function Specification	Library Unit
Function Body	Child Of
Function Subunit Specification	Has Specification Unit
Function Subunit Body	
Package Spec	
Package Body	
Package Body Subunit Specification	
Package Body Subunit Body	
Task Specification	
Task Body	
Task Body Subunit Specification	
Task Body Subunit Body	
Generic Package	
Null Object	

Plot 9 Ada Composer - Netmap Layout - Dependent Step Links from TEXT_IO



Plot 10 Ada Composer - Netmap Layout - Zoomed View

Appendix E - Overview of SEE-Ada Version 3

E.1 Introduction

SEE-Ada is a tool for the visualisation of large, complex Ada software systems. It uses computer graphics to provide meaningful, scaleable views of the total software system, including design and code entities, their attributes and relationships. SEE-Ada can assist in a wide range of software engineering tasks including management, development, independent verification and validation, quality assessment, and software maintenance.

Key features of SEE-Ada are:

- Software System Visualisation environment based on the use of an underlying Software Product Model to support multiple-perspective views and information integration.
- Open environment which allows the import and integration of a wide range of information from a variety of project sources.
- Allows the display of project information integrated with structural representations of the software system.
- Provides the ability to customise and adapt information to specific needs.

E.2 System Framework

Figure 7-1 shows the structure of the SEE-Ada environment.

A range of information can be extracted from software project sources and imported into SEE-Ada. Structural information about the system (eg entities and their structural relationships) can be extracted from the Ada source code or obtained directly from an Ada compilation system via a standard ASIS interface using the filter tools provided. Structural design-level detail can also be imported by way of the Structure I/O feature. Data representing software/project attributes is imported into the SEE-Ada environment from external, commercial or locally developed tools. Many types of information can be imported including requirements, configuration management information, product measures, test results and so forth. Structural information provides the structural element of the Software Product Model (SPM). Other information is integrated into the model as attributes of the structural entities.

Architectural views of the software are generated from the structural model and displayed in graphical form. Attribute information can be integrated into the views and used to describe and provide insights into software characteristics.

SEE-Ada is an open system: design information, development history, and other data from CASE tools, development environments, and other sources can be imported into SEE-Ada and displayed in a consistent, and integrated manner.

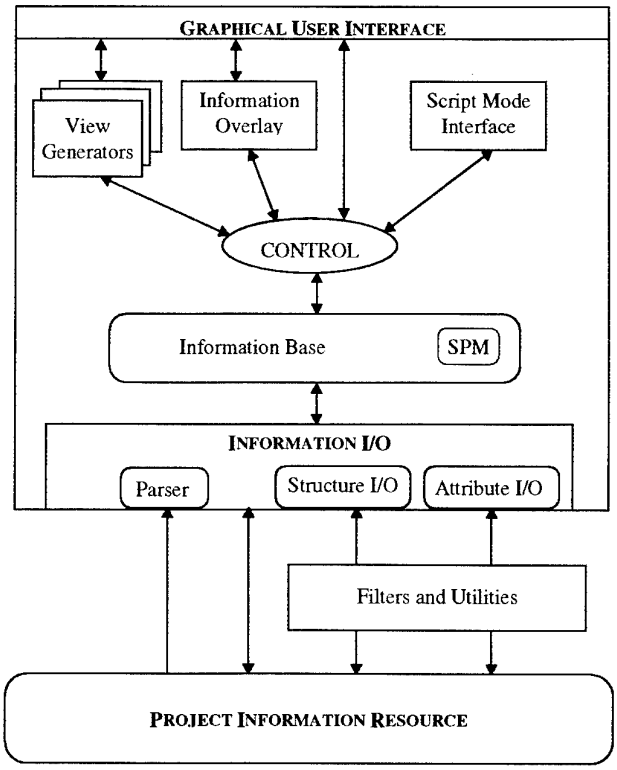


Figure 7-1 The SEE-Ada System Framework

E.3 SEE-Ada Views

Figure 7-2 shows a set of integrated views of a software system as presented by the SEE-Ada product. These views are generated from information captured in the Software Product Model.

The Subsystem View shows 'design-level' information. In this case, the view shows the relationship between logical design entities (class categories filtered from the Rational Rose design tool) and physical design entities (Rational Subsystems from the Rational Apex environment).

The Layers View shows the Ada compilation units which implement the highlighted section of the design shown in the Subsystem View. These Ada units are arranged based on compilation dependencies to provide a compact, spatial representation of the code modules. This view can be customised and tailored to support user needs. The compact representation allows other information to be superimposed. For example, in Figure 7-2, information on the degree of commenting is superimposed via a colour mapping mechanism. The entities shaded red show those source code modules that have no comments. Other colours have been used as threshold values to indicate the degree of commenting.

The Graph View is a view which can be generated from an arbitrary selection of entities. The directed graph representation can display any one of the relationships stored in the Software Product Model. The Graph View shown in Figure 7-2 shows the *with* structure between a selection of packages.

The Contains View shows those subprograms encapsulated in an Ada compilation unit. For example, the TREE_BUILDER package body encapsulates both functions and procedures as shown in Figure 7-2.

The user has selected the "STRING_ASSIGN" procedure and requested a text view to show the related source code. Link attributes associated with the Software Product Model provide the basis for displaying this information.

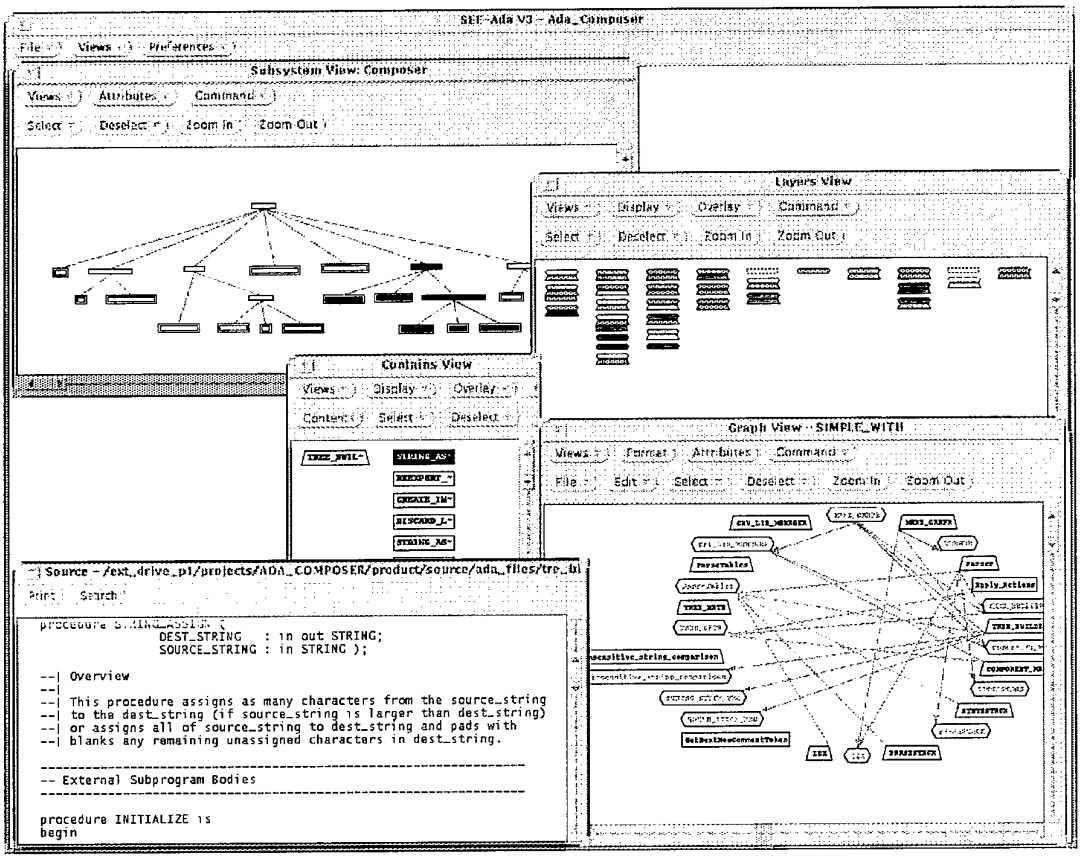


Figure 7-2 : Main Representations Used within SEE-Ada

As can be seen from this figure, a set of integrated views of the software product allows the user to quickly traverse from high-level design concepts to individual lines of code in a consistent way whilst maintaining context.

E.4 Viewing Attribute Information

The integration of information is a key concept that has been explored as part of the AViDeS research. Figure 7-3 shows a Subsystem View and a Layers View with attribute information superimposed.

In this case, attributes were used to identify those entities which declare global variables. The use of global variables can result in highly coupled software which is difficult to maintain. In Ada, the use of global variables in sections of the software which use concurrent threads can result in race conditions. These conditions can

induce serious timing problems and intermittent failures which are difficult to rectify.

Attributes can be overlaid onto any of the Subsystem, Layers, Worksheet, Graph and Contains view via the use of the Attributes Tool as shown in Figure 7-3. The Attributes Tool specifies the mapping of threshold values to up to 5 different colours. This allows both numeric and symbolic data to be overlaid as colours onto the 5 aforementioned views. Individual values can be viewed in a Show Values window if desired.

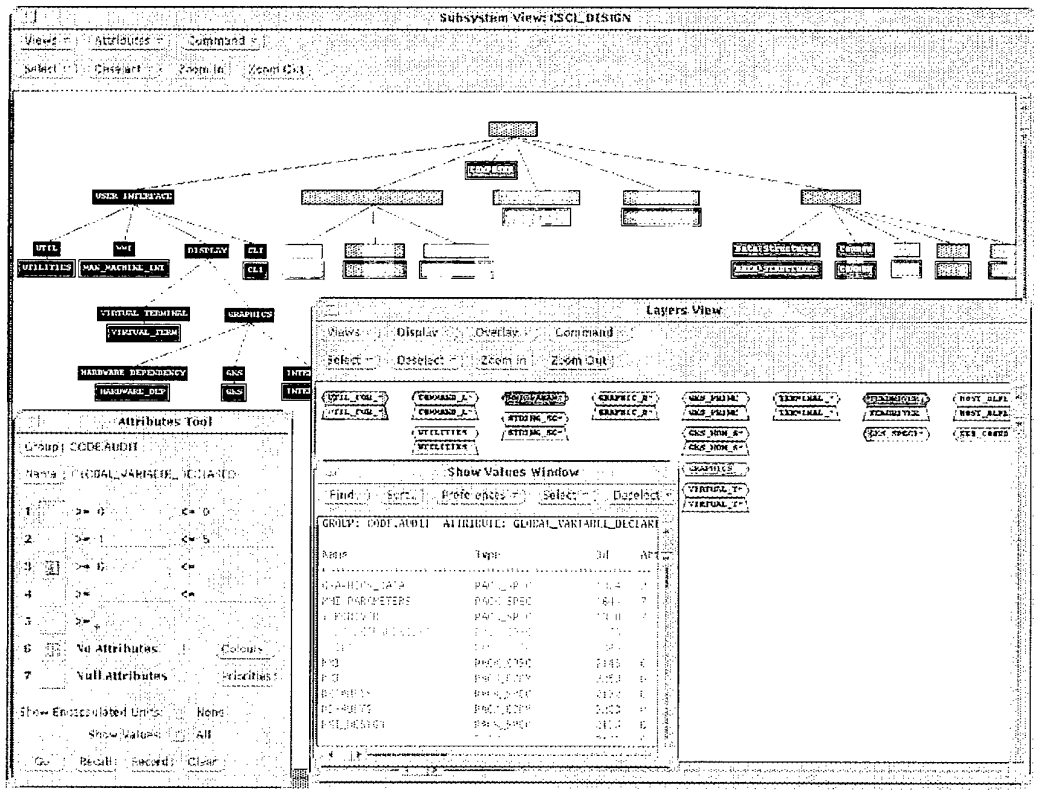


Figure 7-3 Viewing Attributes in SEE-Ada

The approach of integrating information on SEE-Ada views can support a wide range of needs. For example, configuration management information can be used to identify those units that have undergone most change. This information can also be superimposed to show which programmers have authored or changed particular units. Test information can be superimposed to show which units have undergone test, the extent of that testing and the results of particular tests.

E.5 Viewing Relationships

Figure 7-4 shows how SEE-Ada views can be customised to provide required information. Relationships between code entities are typically provided in terms of a directed graph of the complete system (eg the Graph View shows the compilation

dependency 'with' relationship between a subset of units selected in the Layers View). This approach does not scale well and the superfluous information often confuses the user.

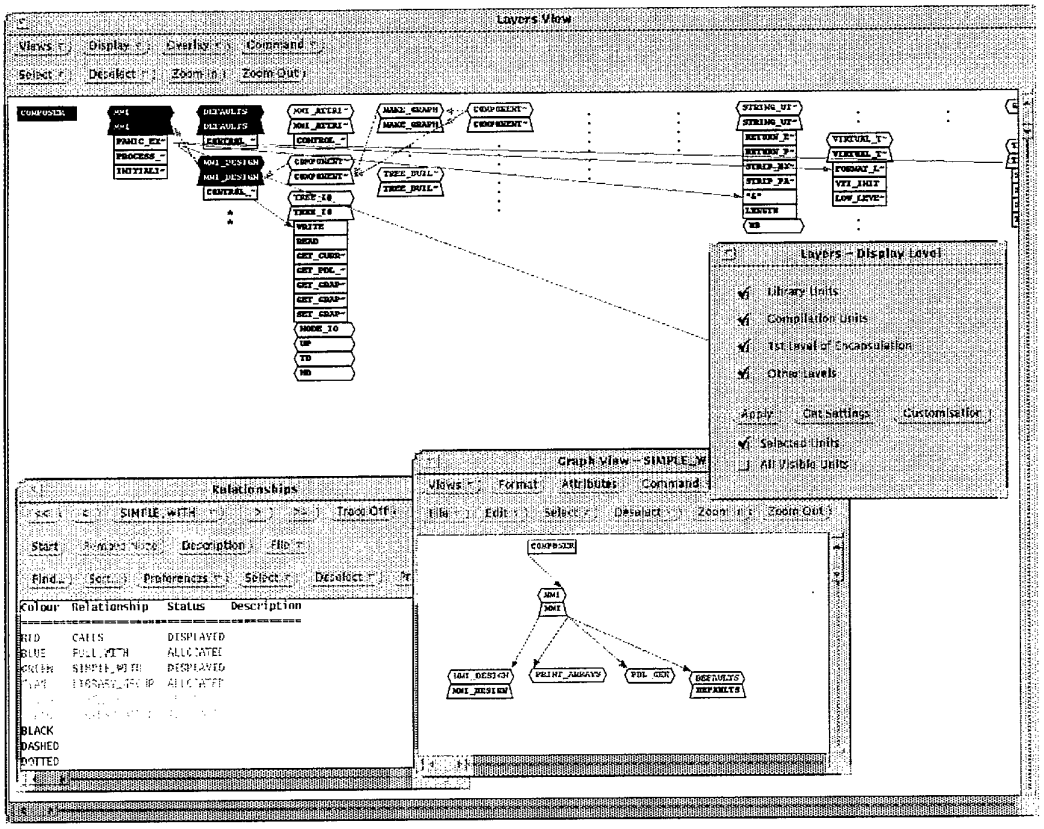


Figure 7-4 Viewing Relationships in SEE-Ada

The integrated visualisation approach as used in SEE-Ada allows the user to query for and superimpose only that information which is necessary for the task at hand. For example, the first level of a call tree from a subprogram has been superimposed in "Red" and the "Green" trace line shows usage of the "COMPONENT_MANAGER" library. Any relationship can be loaded into SEE-Ada. An example of the types of relationships that may be loaded is shown in the Relationships window of Figure 7-4.

A benefit of the integrated visualisation approach is that the information can be customised and adapted for a particular need and the information can be presented in terms of a familiar context (ie the general shape and layout of the compilation unit lattice). The detail level of individual compilation units can be set so as to remove clutter caused by irrelevant information. In Figure 7-4, the detail level of the "TREE_IO" package has been increased to show subprograms. The detail level of other packages has been reduced so that they appear only as points.

Other "secondary views" (eg graph view) can be used to provide supplementary information or present information in a more meaningful way.

E.6 Usage Monitoring

Figure 7-5 shows an example of a SEE-Ada session which has been recorded using the SEE-Ada Usage Monitor. The Usage Monitor captures information about tasks performed with SEE-Ada. When the Usage Monitor is enabled, the 'SEE-Ada Usage Monitor' window is displayed in the top-right corner of the screen. This window shows the current state of the Usage Monitor and also acts as a control panel.

Before using SEE-Ada the user registers the task to be performed in the Setup Window by selecting a task name (usually from a list) and entering a description of the task objective. Other information, such as the current user, is entered automatically, based on the currently logged on user, but may be changed if required. When the OK button is pressed, the Usage Monitor goes into recording mode and the user begins the task.

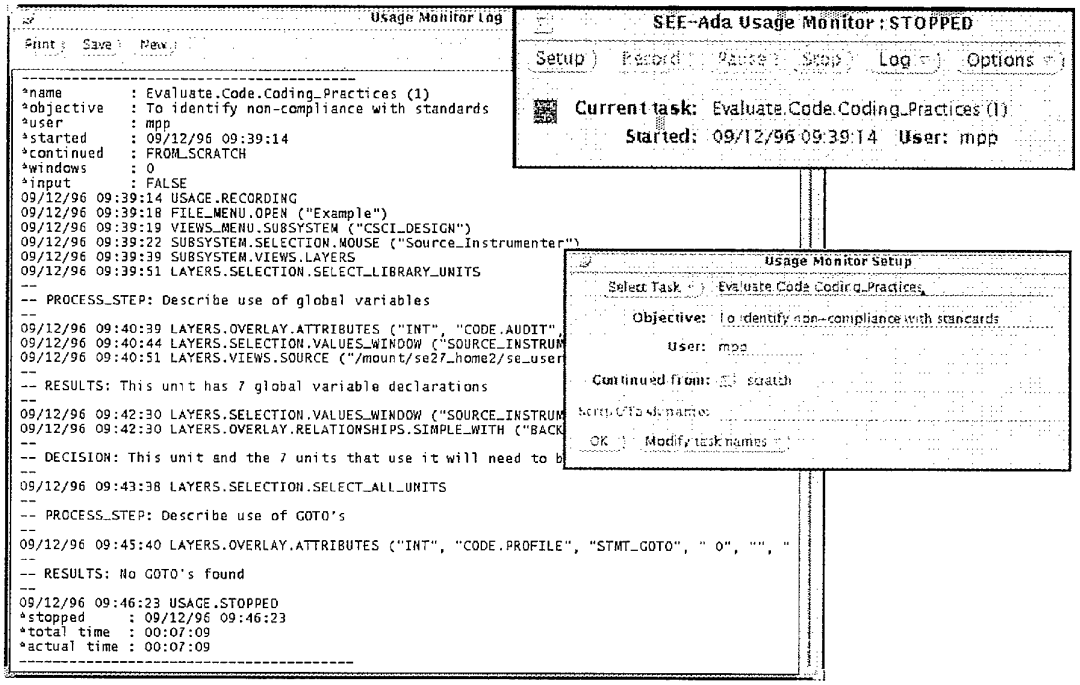


Figure 7-5 SEE-Ada Usage Mode

The Log Window shows the log generated by the Usage Monitor for the current task. The information entered in the Setup Window is recorded in a series of header lines marked by stars. Following the header are the commands executed by the user each marked with the time they were executed. For consistency and readability the commands are represented in the same format used by the SEE-Ada script processor as discussed in Section E.7. Comments can be entered by the user and are preceded by a double dash '--'.

The user completes the task by pressing the stop button on the 'SEE-Ada Usage Monitor' window. This causes the log to be terminated with a footer indicating the total time spent on the task.

E.7 Script Mode

SEE-Ada supports task facilitation by providing a Script Mode Interface. This interface can support the setup of the environment. It also supports the preparation of custom descriptions for particular tasks.

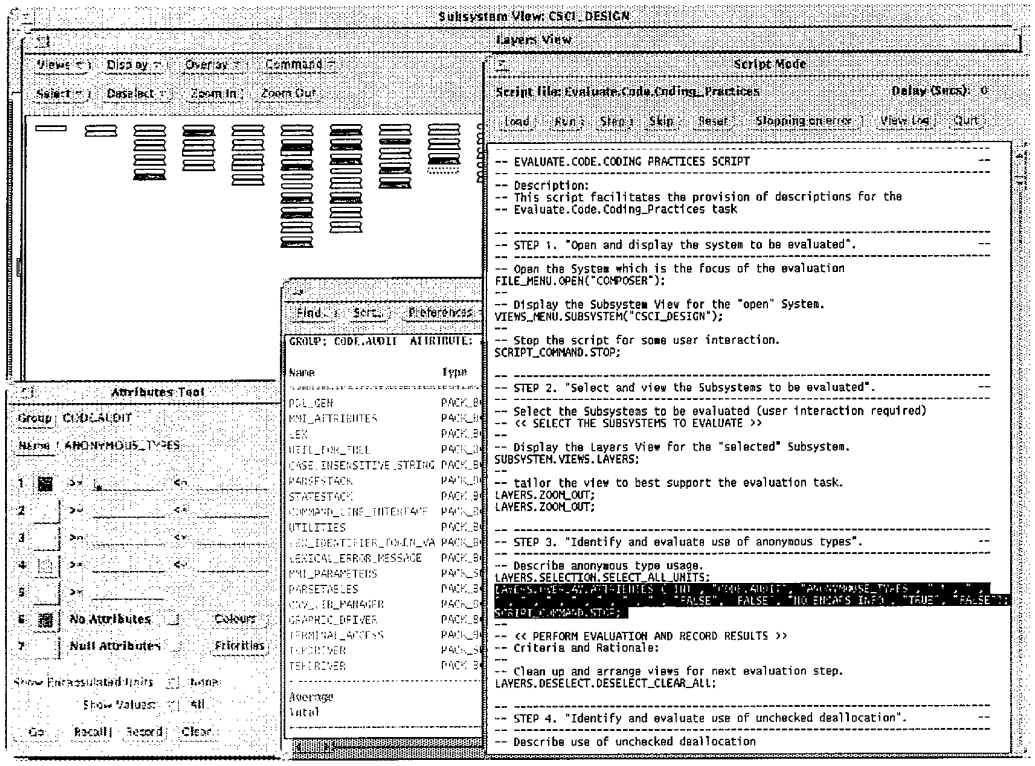


Figure 7-6. SEE-Ada Script Mode

Figure 7-6 provides an example of a script that can be used to support the evaluation of coding practices. The user can either step through each script action or 'Run' the script in which case the script will automatically execute each action until it reaches a 'stop' command. The script begins by opening the system to be evaluated and then automatically displays the Subsystem View called "CSCI_DESIGN". The user then interacts with the environment to select which sections of the system will be evaluated. A Layers View showing the Ada Compilation units for this section of the system is then displayed. The view is then tailored to provide a compact representation onto which other information can be superimposed. The script then supports various evaluation activities. The first phase of the evaluation (at Step 3) is to check for usage of anonymous types. The script selects an attribute that will indicate the use of anonymous types and overlays this information on the Layers View. The user gets an immediate visual indication of whether the code complies with this criterion. The user can then interact with the environment to conduct a further analysis. For example, the user may wish to see how the feature is used in the actual source code or may wish to change the colour mappings to highlight units with the highest proportion of non-compliances.

The user then moves on to the next stage of the evaluation by running or stepping through the script. The script automatically 'cleans up' the views and then produces a visual description which will support the next stage of the evaluation.

By using scripts in this manner, the set of actions that need to be undertaken for these types of evaluation tasks can be recorded and enacted. Relevant descriptions are provided to support the analyst. Although custom descriptions are produced, these can be adapted by users to help support their specific information needs.

Software System Visualisation: Netmap Investigations

Peter Duffett and Rudi Vernik

DISTRIBUTION LIST

Number of Copies

AUSTRALIA

DEFENCE ORGANISATION

Task sponsor:

Head, Systems Acquisition(ES)	1

S&T Program

Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control sheet
Counsellor, Defence Science, Washington	Doc Control sheet
Scientific Adviser to MRDC Thailand	Doc Control sheet
Director General Scientific Advisers and Trials)	1 shared copy
Scientific Adviser - Policy and Command)	
Navy Scientific Adviser	1 copy of Doc Control sheet and 1 distribution list
Scientific Adviser - Army	Doc Control sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1

Aeronautical & Maritime Research Laboratory

Director	1

Electronics and Surveillance Research Laboratory

Director	1
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control sheet
Head, Information Architectures Group	Doc Control sheet

Head, Information Warfare Studies Group	Doc Control sheet
Head, Software Systems Engineering Group	1
Head, Trusted Computer Systems Group	Doc Control sheet
Head, Advanced Computer Capabilities Group	Doc Control sheet
Head, Computer Systems Architecture Group	Doc Control sheet
Head, Systems Simulation and Assessment Group	Doc Control sheet
Head, Intelligence Systems Group	Doc Control sheet
Head, CCIS Interoperbility Lab	Doc Control sheet
Head Command Support Systems Group	Doc Control sheet
Head, C3I Operational Analysis Group	Doc Control sheet
Head Information Management and Fusion Group	Doc Control sheet
Head, Human Systems Integration Group	Doc Control sheet
Task Manager	1
Author	4
Publications and Publicity Officer, ITD	1
DSTO Library and Archives	
Library Fishermens Bend	1
Library Maribyrnong	1
Library Salisbury	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control sheet
Capability Development Division	
Director General Maritime Development	Doc Control sheet
Director General Land Development	Doc Control sheet
Director General C3I Development	Doc Control sheet
Intelligence Program	
Defence Intelligence Organisation	1
Library, Defence Signals Directorate	Doc Control sheet
Acquisition and Logistics Program	
Head, Industry and Procurement Infrastructure	Doc Control sheet
Head, Systems Acquisition (Aerospace)	Doc Control sheet
Head, Systems Acquisition (Maritime and Ground)	Doc Control sheet
Corporate Support Program (libraries)	
OIC TRS Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	1
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1
Universities and Colleges	
Australian Defence Force Academy Library	1
Head of Aerospace and Mechanical Engineering	1

OUTSIDE AUSTRALIA

Abstracting and Information Organisations	
INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Documents Librarian, The Center for Research Libraries, US	1
Information Exchange Agreement Partners	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	5
Total number of copies:	**45**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Software System Visualisation: Netmap Investigations

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Peter Duffett and Rudi Vernik

5. CORPORATE AUTHOR

Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA 5108

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0558	AR-010-284	Technical Report	July 1997

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
9505-13-47	840741	FASDM	65	14

13. DOWNGRADING/DELIMITING INSTRUCTIONS

To be reviewed three years after date of publication

14. RELEASE AUTHORITY

Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Visualization, Software Tools, Systems Engineering

19. ABSTRACT

Defence systems have become increasingly reliant on software. The intangible and complex nature of software makes it difficult to manage and understand. Computer based visualisations of software have shown promise for providing the necessary visibility to acquire, develop, and maintain software systems. In this report we investigate a generic visualisation tool, Netmap, as a means of addressing these visualisation problems. Issues of using generic visualisation tools to support software tasks are discussed.